wxWidgets tutorial

本文来自 http://zetcode.com/gui/wxwidgets/

This is wxWidgets tutorial for the C++ programming language. wxWidgets is a cross platform toolkit or framework for creating C++ GUI applications. After reading this tutorial, you will be able to program non trivial wxWidgets applications.

This is wxWidgets tutorial for the C++ programming language. wxWidgets is a cross platform toolkit or framework for creating C++ GUI applications. After reading this tutorial, you will be able to program non trivial wxWidgets applications.

Introduction ... 3
- wxWidgets ... 3
- The C++ programming language ... 3
- Programming languages ... 3
- Multiplatform programming ... 4

wxWidgets helper classes ... 6
- Console ... 6
- wxString ... 6
- Utility functions ... 10
- Time & date ... 12
- Files ... 15

First programs in wxWidgets ... 20
- A simple application ... 20
- Application icon ... 22
- A simple button ... 25
- Widgets communicate ... 28

Menus and Toolbars ... 34
- Simple menu example ... 34
- Submenus ... 37
- Toolbars ... 40

Layout management in wxWidgets ... 47
- Absolute Positioning ... 47
- Using sizers ... 50
- wxBoxSizer ... 53
- Go To Class ... 59
- wxGridSizer ... 64
- wxFlexGridSizer ... 67

Events ... 71
- Definitions ... 71

A simple event example....71
Event table....72
Example using Connect()....74
Event propagation....76
Vetoing events....79
Window identifiers....82
Dialogs....85
Predefined dialogs....85
Message dialogs....85
wxFileDialog....89
wxFontDialog....92
A custom dialog....95
Widgets....99
wxCheckBox....99
wxBitmapButton....102
wxToggleButton....105
wxStaticLine....110
wxStaticText....113
wxSlider....115
Widgets II....120
wxListBox....120
wxNotebook....126
wxScrolledWindow....130
Drag and Drop....133
Device Contexts....139
Simple line....140
Drawing text....143
Point....145
Pen....148
Regions....151
Gradient....154
Shapes....157
Custom widgets....160
The Burning Widget....160
The Tetris game in wxWidgets....168
The development....168

Introduction

This is a tutorial, that will introduce you to the programming with the wxWidgets toolkit.

wxWidgets

wxWidgets is a GUI (Graphical User Interface) toolkit for creating C++ applications. It is an opensource, mature and cross-platform toolkit. wxWidgets applications run on all major OS platforms, Windows, Unix and Mac. The project was started by **Julian Smart** in 1992. wxWidgets is much more than a toolkit. It provides a large variety of classes for handling streams, databases, threads, online help or application settings. wxWidgets consists of a large group of widgets. The community around wxWidgets is grouped around their website.

The C++ programming language

The C++ programming language is one of the most widely used programming languages. It is used in many famous desktop applications like MS Office, Macromedia Flash, Firefox, Photoshop or 3D Max. C++ dominates also the world of PC games. It is one of the most difficult programming languages. On the other hand, programming in C++ today is different from programming 10 years ago. There are many tools and libraries that make the programming easier.

Programming languages

There are currently several widely used programming languages. The following list is based on theTIOBE Programming Community Index. The numbers are from November 2010. As we can see, C++ still belongs to the most popular programming languages in the world.

Position	Language	Ratings
1	Java	18.5%

2	C	16.7%
3	C++	9.5%
4	PHP	7.8%
5	C#	5.7%
6	Python	5.7%
7	Visual Basic	5.5%
8	Objective C	3.2%
9	Perl	2.4%
10	Ruby	1.9%

Java is the most widely used programming language. Java excels in creating portable mobile applications, programming various appliances and in creating enterprise applications. Every fourth application is programmed in C/C++. They are standard for creating operating systems and various desktop applications. C/C++ are the most widely used system programming languages.

PHP dominates over the Web. While Java is used mainly by large organizations, PHP is used by smaller companies and individuals. PHP is used to create dynamic web applications.

C# is the main programming language of the Microsoft .NET platform. C# is followed in .NET by Visual Basic. It represents of the popularity of the RAD. (Rapid Application Development.)

Perl, Python and Ruby are the most widely used scripting languages. They share many similarities. They are close competitors.

The Objective C is the main programming language of the Apple ecosystem.

Multiplatform programming

Today, multiplatform programming is a buzzword. Most languages and libraries want to be multiplatform. wxWidgets was created as a multiplatform tool from the beginning. Most developers choose among these options. If it is

possible, they go to the web. Or they can use Qt, wxWidgets, Swing or SWT. The Qt library is the closest competitor to wxWidgets. Using the right tool for the right job is one of the most importat decisions programmer's (or management) do.

This was an introduction to wxWidgets.

wxWidgets helper classes

wxWidgets consists of a large group of helper classes, that help programmers to do their job. These include classes for working with strings, files, XML files, streams, database or network. Here we will show only a tiny drop of the whole lake.

wxWidgets library can be used to create console and gui applications. In this chapter, we will illustrate some of the helper classes in console based applications.

Console

This is a simple console application. The application puts some text into the console window.

console.cpp

```
#include <wx/string.h>


int main(int argc, char **argv)
{
  wxPuts(wxT("A wxWidgets console application"));
}
```

Output.

```
A wxWidgets console application
```

wxString

wxString is a class representing a character string.

In the following example, we define three wxStrings. We create one string of these strings using addition operation.

addition.cpp

```
#include <wx/string.h>


int main(int argc, char **argv)
{
  wxString str1 = wxT("Linux");
  wxString str2 = wxT("Operating");
  wxString str3 = wxT("System");


  wxString str = str1 + wxT(" ") + str2 + wxT(" ") + str3;


  wxPuts(str);
}
```

Output.

```
Linux Operating System
```

The Printf() method is used to format strings.

formatted.cpp

```
#include <wx/string.h>


int main(int argc, char **argv)
```

```
{


  int flowers = 21;


  wxString str;
  str.Printf(wxT("There are %d red roses."), flowers);
  //上面的语句不输出，只是通过格式
  wxPuts(str);
}
```

Output.

```
There are 21 red roses.
```

The following example checks, whether a string contains another string. For this we have a*Contains()* method.

contains.cpp

```
#include <wx/string.h>


int main(int argc, char **argv)
{


  wxString str = wxT("The history of my life");


  if (str.Contains(wxT("history"))) {
      wxPuts(wxT("Contains!"));
  }
```

```
  if (!str.Contains(wxT("plain"))) {
      wxPuts(wxT("Does not contain!"));
  }
}
```

Output.

```
Contains!
Does not contain!
```

The *Len()* method returns the number of characters in the string.

length.cpp

```
#include <wx/string.h>


int main(int argc, char **argv)
{
  wxString str = wxT("The history of my life");
  wxPrintf(wxT("The string has %d characters\n"), str.Len());
  //前面例子中是成员函数 Prinft()
}
```

Output.

```
The string has 22 characters
```

The *MakeLower()* and *MakeUpper()* methods make characters lower case and upper case.

cases.cpp

```
#include <wx/string.h>


int main(int argc, char **argv)
{
  wxString str = wxT("The history of my life");


  wxPuts(str.MakeLower());
  wxPuts(str.MakeUpper());
}
```

Output.

```
the history of my life
THE HISTORY OF MY LIFE
```

Utility functions

wxWidgets has several handy utility functions for executing a process, getting a home user directory or getting the OS name.

In the following example, we execute the ls command. For this, we have the wxShell() function. Unix only.

shell.cpp

```
#include <wx/string.h>
#include <wx/utils.h>
```

```
int main(int argc, char **argv)
{

  wxShell(wxT("ls -l"));  //类似于 System()函数

}
```

Output.

```
total 40
-rwxr-xr-x 1 vronskij vronskij  9028 2007-09-06 22:10 basic
-rw-r--r-- 1 vronskij vronskij    95 2007-09-06 22:09 basic.cpp
-rw-r--r-- 1 vronskij vronskij   430 2007-09-06 00:07 basic.cpp~
-rwxr-xr-x 1 vronskij vronskij 11080 2007-09-05 23:17 console
-rw-r--r-- 1 vronskij vronskij   500 2007-09-05 23:17 console.cpp
-rw-r--r-- 1 vronskij vronskij   485 2007-09-05 23:16 console.cpp~
```

Next we will we will get the home user directory, os name, user name, host name and total free memory.

system.cpp

```
#include <wx/string.h>
#include <wx/utils.h>

int main(int argc, char **argv)
{
  wxPuts(wxGetHomeDir());
```

```
  wxPuts(wxGetOsDescription());
  wxPuts(wxGetUserName());
  wxPuts(wxGetFullHostName());

  long mem = wxGetFreeMemory().ToLong();

  wxPrintf(wxT("Memory: %ld\n"), mem);
}
```

Output.

```
/home/vronskij
Linux 2.6.20-16-generic i686
jan bodnar
spartan
Memory: 741244928
```

Time & date

In wxWidgets, we have several classes for working with date & time.

The example shows current date or time in various formats.

datetime.cpp

```
#include <wx/datetime.h>

int main(int argc, char **argv)
{
```

```
  wxDateTime now = wxDateTime::Now();


  wxString date1 = now.Format(); //日期+时间
  wxString date2 = now.Format(wxT("%X"));//仅时间
  wxString date3 = now.Format(wxT("%x"));//仅日期


  wxPuts(date1);
  wxPuts(date2);
  wxPuts(date3);
}
```

Output.

```
Fri Sep  7 21:28:38 2007
21:28:38
09/07/07
```

Next we will show current time in different cities.

datetime2.cpp

```
#include <wx/datetime.h>
//这个程序有运行错?
int main(int argc, char **argv)
{
  wxDateTime now = wxDateTime::Now();


  wxPrintf(wxT("   Tokyo: %s\n"), now.Format(wxT("%a %T"),
      wxDateTime::GMT9).c_str());
```

```
  wxPrintf(wxT("  Moscow: %s\n"), now.Format(wxT("%a %T"),
      wxDateTime::MSD).c_str());
  wxPrintf(wxT("Budapest: %s\n"), now.Format(wxT("%a %T"),
      wxDateTime::CEST).c_str());
  wxPrintf(wxT("  London: %s\n"), now.Format(wxT("%a %T"),
      wxDateTime::WEST).c_str());
  wxPrintf(wxT("New York: %s\n"), now.Format(wxT("%a %T"),
      wxDateTime::EDT).c_str());
}
```

Output.

```
   Tokyo: Sat 05:42:24
  Moscow: Sat 00:42:24
Budapest: Fri 22:42:24
  London: Fri 22:42:24
New York: Fri 16:42:24
```

The following example shows, how we can add date spans to our date/time. We add one month to the current time.

datespan.cpp

```
#include <wx/datetime.h>


int main(int argc, char **argv)
{
  wxDateTime now = wxDateTime::Now();
  wxString date1 = now.Format(wxT("%B %d %Y"));
```

```
  wxPuts(date1);


  wxDateSpan span(0, 1);
  wxDateTime then = now.Add(span);


  wxString date2 = then.Format(wxT("%B %d %Y"));
  wxPuts(date2);
}
```

Output.

```
September 07 2007
October 07 2007
```

Files

wxWidgets has several classes to facilitate working with files. This is low level access to files, as opposed to working with streams.

In the following example, we use the **wxFile** class to create a new file and write data to it. We also test, whether the file is opened. Note, that when we create a file, it automatically stays as opened.

createfile.cpp

```
#include <wx/file.h>


int main(int argc, char **argv)
{
```

```
  wxString str = wxT("You make me want to be a better man.\n");


  wxFile file;
  file.Create(wxT("quote"), true);


  if (file.IsOpened())
      wxPuts(wxT("the file is opened"));


  file.Write(str);
  file.Close();


  if (!file.IsOpened())
      wxPuts(wxT("the file is not opened"));
}
```

Output.

```
$ ls qoute
ls: qoute: No such file or directory


$ ./createfile
the file is opened
the file is not opened


$ cat quote
You make me want to be a better man.
```

The **wxTextFile** is a simple class which allows to work with text files on line by line basis. It is easier to work with this class than with wxFile class.

In the next example, we will print the number of lines in a file, first and last lines and finally we will read and show the contents of the file.

readfile.cpp

```
#include <wx/textfile.h>


int main(int argc, char **argv)
{
  //运行前，请自行建 test.c 文本文件
  wxTextFile file(wxT("test.c"));


  file.Open();


  wxPrintf(wxT("Number of lines: %d\n"), file.GetLineCount());
  wxPrintf(wxT("First line: %s\n"), file.GetFirstLine().c_str());
  wxPrintf(wxT("Last line: %s\n"), file.GetLastLine().c_str());


  wxPuts(wxT("-------------------------------------"));


  wxString s;


  for ( s = file.GetFirstLine(); !file.Eof();
      s = file.GetNextLine() )
  {
       wxPuts(s);
  }


  file.Close();
```

```
}
```

Output.

```
Number of lines: 8

First line: #include Last line: } ------------------------------------
#include <glib.h> #include <glib/gstdio.h> int main()
{ g_mkdir("/home/vronskij/test", S_IRWXU); }
```

The **wxDir** class allows us to enumerate files and directories.

In the following example, we will print all files and directories available in the current working directory.

dir.cpp

```
#include <wx/dir.h>

#include <wx/filefn.h>


int main(int argc, char **argv)

{


  wxDir dir(wxGetCwd());


  wxString file;


  bool cont = dir.GetFirst(&file, wxEmptyString,

      wxDIR_FILES | wxDIR_DIRS);


  while (cont) {

      wxPuts(file);

      cont = dir.GetNext(&file);
```

```
  }
}
```

Output.

```
$ ./dir
dir
temp
console
basic.cpp
basic
quote
createfile
console.cpp
basic.cpp~
test.c
console.cpp~
```

In this chapter, we have covered some wxWidgets helper classes.

First programs in wxWidgets

In this chapter, we will cover the basics needed to create wxWidgets applications. We will create our first simple example, show how to display an icon. Next we will create a simple example demonstrating usage of an event. Finally, we will see, how widgets communicate in wxWidgets applications.

A simple application

First we create the very basic wxWidgets program.

simple.h

```
#include <wx/wx.h>

class Simple : public wxFrame
{
public:
    Simple(const wxString& title);
};
```

simple.cpp

```
#include "simple.h"

Simple::Simple(const wxString& title)
       : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(250,
150))
{
  Centre();
}
```

main.h

```
#include <wx/wx.h>
```

```
class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "simple.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{
    Simple *simple = new Simple(wxT("Simple"));
    simple->Show(true);

    return true;
}
```

This very basic example shows a small window on the screen. The window is centered.

```
Centre();
```

This method centers the window on the screen. Both horizontally and vertically.

```
IMPLEMENT_APP(MyApp)
```

The code that implements the application is hidden behind this macro. This is copy and paste code, we usually don't have to care about.

IMPLEMENT_APP 是 wxWindgets 定义的宏，Note the use of IMPLEMENT_APP(appClass), which allows wxWidgets to dynamically create an instance of the application object at the appropriate point in wxWidgets

initialization.

To compile the example, run the following command. (On Unix).

```
g++ main.cpp main.h simple.cpp simple.h  `wx-config --cxxflags --libs`
-o simple
```

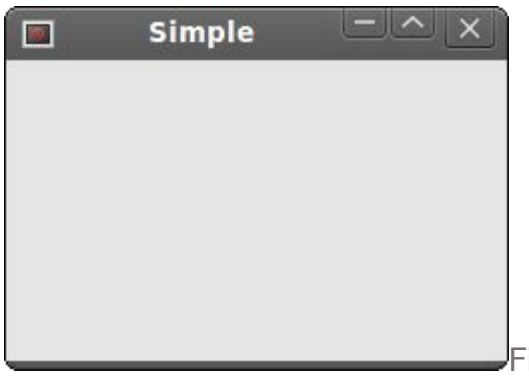


Figure: Simple

Application icon

In this example, we provide an icon for our application. It became a standard to display a small icon in the upper left corner of the window. The icon is a graphical identity of the program.

icon.h

```
#include <wx/wx.h>

class Icon : public wxFrame
{
public:
    Icon(const wxString& title);
};
```

icon.cpp

```
#include "icon.h"

Icon::Icon(const wxString& title)
      : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(250,
150))
```

```
{
  SetIcon(wxIcon(wxT("web.xpm")));
  Centre();
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "icon.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{
    Icon *icon = new Icon(wxT("Icon"));
    icon->Show(true);

    retun true;
}
```

In our example we show a small web icon.

```
SetIcon(wxIcon(wxT("web.xpm")));
```

To display an application icon is a matter of one code line. XPM (X PixMap) is an ASCII image format.

Figure: Icon

.xpm 格式的文件，XPM(XPixMap)图形格式在 X11中是一个标准格式，它把图形保存成 ASCII 文本，一个 XPM 的定义不仅仅是 ASCII 形式，它的格式还可以是 C 源代码形式的，可以直接将它编辑到自己的应用程序中去。

下面是 wxWidgets 3.0.0 Demo 中一个图片的定义：

```
/* XPM */
static const char *const bombs_xpm[] = {
/* columns rows colors chars-per-pixel */
"32 32 6 1",      //这幅图像有32行32列，共6种颜色，每像占一个字节
"  c Black",      //空格代表颜色 Black
". c Blue",      //.代表颜色 Blue
"X c #00bf00",  //X 代表颜色#00bf00
"o c Red",
"O c Yellow",
"+ c Gray100",
/* pixels */
"                               ",    //从这一行开始是像素颜色的描述了
" oooooo ++++++++++++++++++++++ ",
" oooooo ++++++++++++++++++++++ ",
" oooooo ++++++++++++++++++++++ ",
" oooooo ++++++++++++++++++++++ ",
" oooooo ++++++++++++++++++++++ ",
" oooooo ++++++++++++++++++++++ ",
" oooooo ++++++++++++++++++++++ ",
"                               ",
" ++++++ ++++++++++++++++++ .... ",
" ++++++ ++++++++++++++++++ .... ",
" ++++++ ++++++++++++++++++ .... ",
" ++++++ ++++++++++++++++++ .... ",
" ++++++ ++++++++++++++++++ .... ",
" ++++++ ++++++++++++++++++     ",
" ++++++ ++++++++++++++++++ ++++ ",
```

```
" ++++++ ++++++++++++++++++ ++++ ",
" ++++++ ++++++++++++++++++ ++++ ",
" ++++++ ++++++++++++++++++ ++++ ",
" ++++++ ++++++++++++++++++ ++++ ",
" ++++++ ++++++++++++++++++ ++++ ",
" ++++++ ++++++++++++++++++ ++++ ",
" ++++++ ++++++++++++++++++ ++++ ",
" ++++++ ++++++++++++++++++ ++++ ",
" ++++++                   ++++ ",
" ++++++ OOOOOOOOOOOO XXXXX ++++ ",
" ++++++ OOOOOOOOOOOO XXXXX ++++ ",
" ++++++ OOOOOOOOOOOO XXXXX ++++ ",
" ++++++ OOOOOOOOOOOO XXXXX ++++ ",
" ++++++ OOOOOOOOOOOO XXXXX ++++ ",
" ++++++ OOOOOOOOOOOO XXXXX ++++ ",
"                            "
};
```

A simple button

In the following example, we create a button on the frame widget. We will show, how to create a simple event handler.

button.h

```
#include <wx/wx.h>

class Button : public wxFrame
{
public:
    Button(const wxString& title);

    void OnQuit(wxCommandEvent & event);
};
```

button.cpp

```
#include "button.h"
```

```
Button::Button(const wxString& title)
       : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(270,
150))
{
  wxPanel *panel = new wxPanel(this, wxID_ANY);

  wxButton *button = new wxButton(panel, wxID_EXIT, wxT("Quit"),
      wxPoint(20, 20));   //按钮的 ID 为 wxID_EXIT
  Connect(wxID_EXIT, wxEVT_COMMAND_BUTTON_CLICKED,
      wxCommandEventHandler(Button::OnQuit));  //将 wxID_EXIT 按钮的
wxEVT_COMMAND_BUTTON_CLICKED 事件同 Button::OnQuit 联系在一起，当在按钮上
的单击事件发生时，将执行 Button::OnQuit
  button->SetFocus();
  Centre();
}

void Button::OnQuit(wxCommandEvent & WXUNUSED(event))
{
    Close(true);
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "button.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{
```

```
    Button *btnapp = new Button(wxT("Button"));
    btnapp->Show(true);

    return true;
}
```

```
wxPanel *panel = new wxPanel(this, wxID_ANY);
```

First we create a *wxPanel* widget. It will be placed inside a *wxFrame* widget.

```
wxButton *button = new wxButton(panel, wxID_EXIT, wxT("Quit"),
wxPoint(20, 20));
```

We create a *wxButton* widget. It is placed on the panel. We use the predefined *wxID_EXIT* id for the button. It will cause to display a small exit icon on the button. The label of the button is "Quit". The button is positioned manually at x=20, y=20 coordinates. The beginning of the coordinate system is at the upper left hand corner.

```
Connect(wxID_EXIT, wxEVT_COMMAND_BUTTON_CLICKED,
    wxCommandEventHandler(Button::OnQuit));
```

If we click on the button, a *wxEVT_COMMAND_BUTTON_CLICKED* event is generated. We connect the event to the *OnQuit()* method of the Button class. So when we click on the button, the OnQuit() method is called.

```
button->SetFocus();
```

We set the keyboard focus to the button. So if we press the enter key, the button is being clicked.

```
Close(true);
```

Inside the OnQuit() method, we call the *Close()* method. This will terminate our application.

Figure: Button

Widgets communicate

It is important to know, how widgets can communicate in application. Follow the next example.

Panels.h

```
#include <wx/wx.h>
#include <wx/panel.h>

class LeftPanel : public wxPanel   //左侧面板，设置按钮等
{
public:
    LeftPanel(wxPanel *parent);

    void OnPlus(wxCommandEvent & event);
    void OnMinus(wxCommandEvent & event);

    wxButton *m_plus;    //加按钮
    wxButton *m_minus;   //减按钮
    wxPanel *m_parent;   //父面板
    int count;

};

class RightPanel : public wxPanel   //右面板，用于显示左面板类的 count
{
public:
    RightPanel(wxPanel *parent);
```

```
    void OnSetText(wxCommandEvent & event);

    wxStaticText *m_text;   //文本框，用于显示 count
};

const int ID_PLUS = 101;
const int ID_MINUS = 102;
```

Panels.cpp

```
#include <wx/stattext.h>
#include "Communicate.h"

LeftPanel::LeftPanel(wxPanel * parent)
      : wxPanel(parent, -1, wxPoint(-1, -1), wxSize(-1, -1),
wxBORDER_SUNKEN)
{
  count = 0;
  m_parent = parent;
  m_plus = new wxButton(this, ID_PLUS, wxT("+"),
      wxPoint(10, 10));   //新建+按钮
  m_minus = new wxButton(this, ID_MINUS, wxT("-"),
      wxPoint(10, 60));  //新建-按钮
  Connect(ID_PLUS, wxEVT_COMMAND_BUTTON_CLICKED,
      wxCommandEventHandler(LeftPanel::OnPlus));
  Connect(ID_MINUS, wxEVT_COMMAND_BUTTON_CLICKED,
      wxCommandEventHandler(LeftPanel::OnMinus));
}

void LeftPanel::OnPlus(wxCommandEvent & WXUNUSED(event))
{
  count++;

  Communicate *comm = (Communicate *) m_parent->GetParent();
//Communicate 见后面的定义，该类中包括左、右两个面板成员，comm 是左面板的父
frame
  comm->m_rp->m_text->SetLabel(wxString::Format(wxT("%d"), count));
//通过父 frame 引用右面板的 m_text
}

void LeftPanel::OnMinus(wxCommandEvent & WXUNUSED(event))
```

```
{
  count--;

  Communicate *comm = (Communicate *) m_parent->GetParent();
  comm->m_rp->m_text->SetLabel(wxString::Format(wxT("%d"), count));
}

RightPanel::RightPanel(wxPanel * parent)
       : wxPanel(parent, wxID_ANY, wxDefaultPosition,
         wxSize(270, 150), wxBORDER_SUNKEN)
{
    m_text = new wxStaticText(this, -1, wxT("0"), wxPoint(40, 60)); //
与左面板中的 count=0对应
}
```

Communicate.h

```
#include "Panels.h"
#include <wx/wxprec.h>

class Communicate : public wxFrame
{
public:
    Communicate(const wxString& title);

    LeftPanel *m_lp;
    RightPanel *m_rp;
    wxPanel *m_parent;
};
```

Communicate.cpp

```
#include "Communicate.h"

Communicate::Communicate(const wxString& title)
       : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(290,
150))
{
  m_parent = new wxPanel(this, wxID_ANY);

  wxBoxSizer *hbox = new wxBoxSizer(wxHORIZONTAL);
```

```
  m_lp = new LeftPanel(m_parent);  //建左面板
  m_rp = new RightPanel(m_parent);  //建右面板

  hbox->Add(m_lp, 1, wxEXPAND | wxALL, 5);
  hbox->Add(m_rp, 1, wxEXPAND | wxALL, 5);

  m_parent->SetSizer(hbox);

  this->Centre();
}
```

注：wxBoxSizer 是 wxSize 的派生类，是可变大小的控件“容器”，用于在其中放其他控件。Add 方法的功能是在其中加入子对象，参数如下：

```
wxSizerItem* wxSizer::Add     (
               wxWindow*      window,  //要加入的窗口
               int     proportion= 0,  //是否可改变大小
               int     flag= 0,   //或运算起来，影响其行为
               int     border= 0,  //边框宽度
               wxObject*      userData= NULL //附加对象，一般不用
       )
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "Communicate.h"

IMPLEMENT_APP(MyApp)
```

```
bool MyApp::OnInit()
{
    Communicate *communicate = new Communicate(wxT("Widgets
communicate"));
    communicate->Show(true);

    return true;
}
```

In our example we have two panels. A left and right panel. The left panel has two buttons. The right panel has one static text. The buttons change the number displayed in the static text. The question is, how do we grab the pointer to the static text?

```
m_parent = parent;
```

Here we save the pointer to the parent widget of the LeftPanel. It is a wxPanel `widget`.

```
Communicate *comm = (Communicate *) m_parent->GetParent();
comm->m_rp->m_text->SetLabel(wxString::Format(wxT("%d"), count));
```

These two lines are the most important lines of the example. It is shown, how we get access to the static text widget, which is placed on a different panel. First we get the parent of the both left and right panels. This parent widget has a pointer to the right panel. And the right panel has a pointer to the static text.

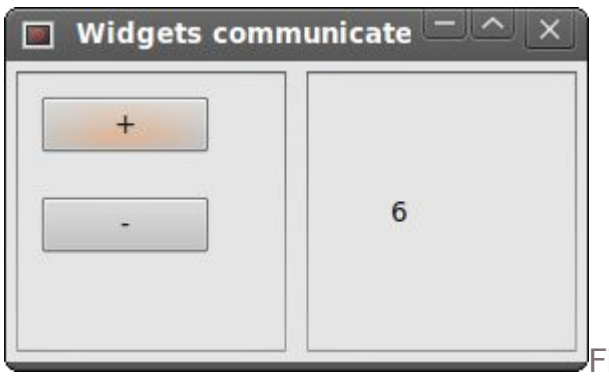
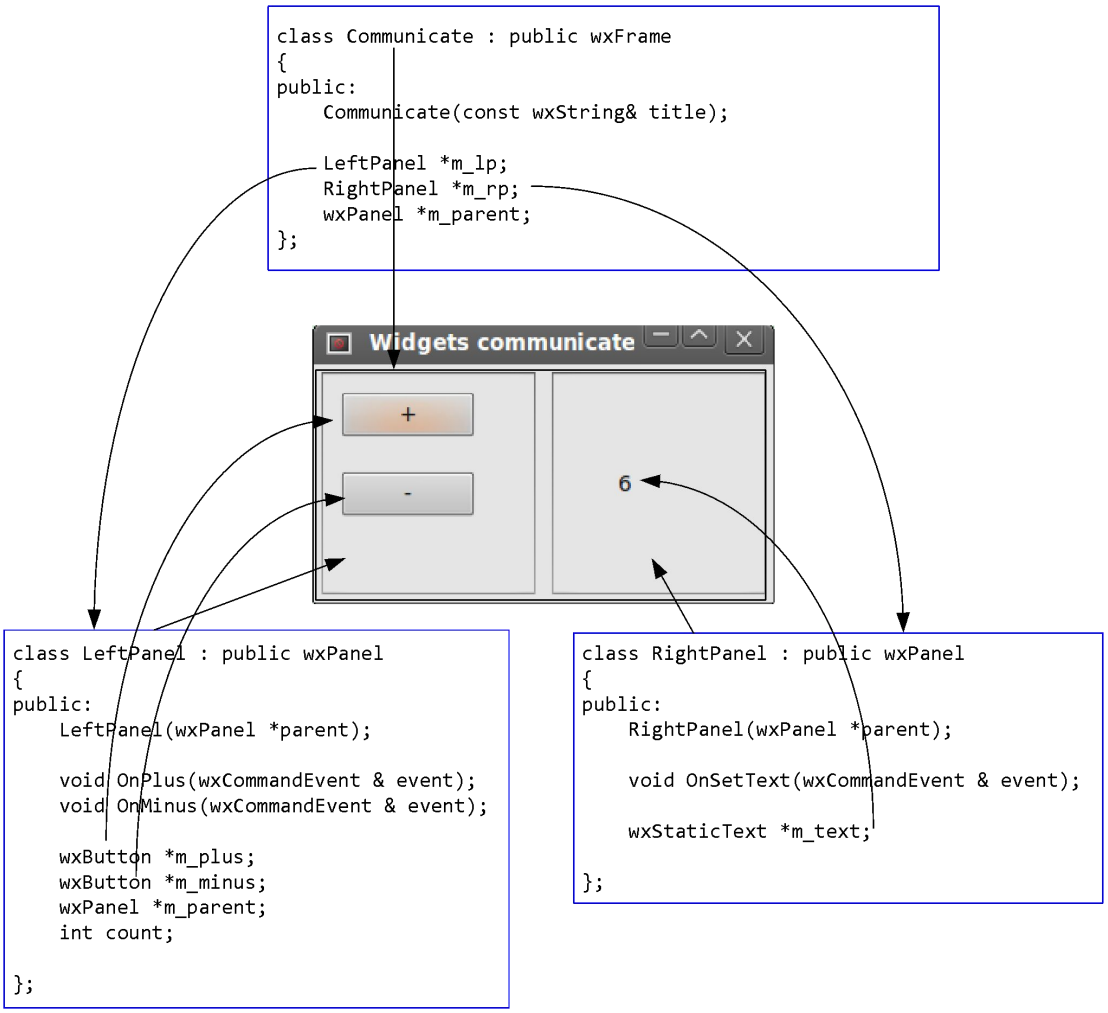


Figure: Widgets communicate

下面结合运行结果看各类之间的关系：



In this part of the wxWidgets tutorial, we have created some simple programs.

Menus and Toolbars

A menubar is one of the most visible parts of the GUI application. It is a group of commands located in various menus. While in console applications you had to remember all those arcane commands, here we have most of the commands grouped into logical parts. There are accepted standards that further reduce the amount of time spending to learn a new application. To implement a menubar in wxWidgets we need to have three things. A wxMenuBar, a wxMenu and a wxMenuItem.

Simple menu example

Creating a menubar in wxWidgets is very simple. Just a few lines of code.

menu.h

```
#include <wx/wx.h>
#include <wx/menu.h>

class SimpleMenu : public wxFrame
{
public:
    SimpleMenu(const wxString& title);

    void OnQuit(wxCommandEvent& event);

    wxMenuBar *menubar;
    wxMenu *file;
};
```

menu.cpp

```
#include "menu.h"

SimpleMenu::SimpleMenu(const wxString& title)
       : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(280,
180))
{
  menubar = new wxMenuBar;
```

```
  file = new wxMenu;
  file->Append(wxID_EXIT, wxT("&Quit"));
  menubar->Append(file, wxT("&File"));
  SetMenuBar(menubar);

  Connect(wxID_EXIT, wxEVT_COMMAND_MENU_SELECTED,
      wxCommandEventHandler(SimpleMenu::OnQuit));
  Centre();
}

void SimpleMenu::OnQuit(wxCommandEvent& WXUNUSED(event))
{
  Close(true);
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "menu.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{
    SimpleMenu *menu = new SimpleMenu(wxT("Simple Menu"));
    menu->Show(true);

    return true;
}
```

```
menubar = new wxMenuBar;
```

First we create a menubar object.

```
file = new wxMenu;
```

Next we create a menu object.

```
file->Append(wxID_EXIT, wxT("&Quit"));
```

We append a menu item into the menu object. The first parameter is the id of the menu item. The second parameter is the name of the menu item. Here we did not create a wxMenuItem explicitly. It was created by the Append() method behind the scenes. Later on, we will create a wxMenuItem manually.

```
menubar->Append(file, wxT("&File"));
SetMenuBar(menubar);
```

After that, we append a menu into the menubar. The & character creates an accelerator key. The character that follows the & is underlined. This way the menu is accessible via the alt + F shortcut. In the end, we call the SetMenuBar() method. This method belongs to the wxFrame widget. It sets up the menubar.

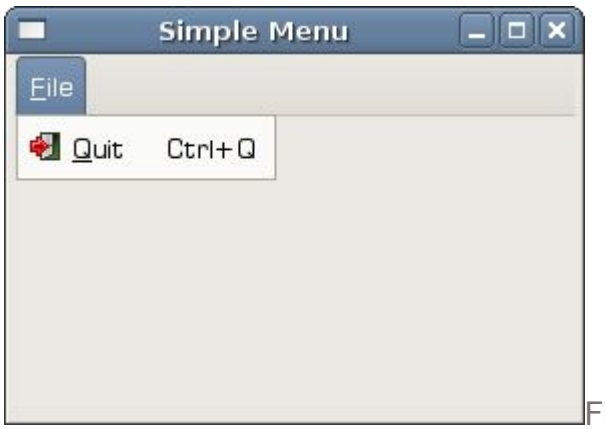


Figure: Simple menu example

Submenus

Each menu can also have a submenu. This way we can group similar commnads into groups. For example we can place commands that hide/show various toolbars like personal bar, address bar, status bar or navigation bar into a submenu called toolbars. Within a menu, we can seperate commands with a separator. It is a simple line. It is common practice to separate commands like new, open, save from commands like print, print preview with a single separator. In our example we will see, how we can create submenus and menu separators.

menu.h

```
#include <wx/wx.h>
#include <wx/menu.h>

class SubMenu : public wxFrame
{
public:
  SubMenu(const wxString& title);

  void OnQuit(wxCommandEvent & event);

  wxMenuBar *menubar;
  wxMenu *file;
  wxMenu *imp;
  wxMenuItem *quit;
};
```

menu.cpp

```
#include "menu.h"

SubMenu::SubMenu(const wxString& title)
       : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(280,
180))
{
  menubar = new wxMenuBar;
  file = new wxMenu;
```

```
  file->Append(wxID_ANY, wxT("&New"));
  file->Append(wxID_ANY, wxT("&Open"));
  file->Append(wxID_ANY, wxT("&Save"));
  file->AppendSeparator();   //加入一条分隔线

  imp = new wxMenu;
  imp->Append(wxID_ANY, wxT("Import newsfeed list..."));
  imp->Append(wxID_ANY, wxT("Import bookmarks..."));
  imp->Append(wxID_ANY, wxT("Import mail..."));

  file->AppendSubMenu(imp, wxT("I&mport"));   //imp 作为 file 的子菜单

  quit = new wxMenuItem(file, wxID_EXIT, wxT("&Quit\tCtrl+W"));
  file->Append(quit);  //wxMenu::Append 有四种重载的形式

  menubar->Append(file, wxT("&File"));
  SetMenuBar(menubar);

  Connect(wxID_EXIT, wxEVT_COMMAND_MENU_SELECTED,
      wxCommandEventHandler(SubMenu::OnQuit));
  Centre();
}

void SubMenu::OnQuit(wxCommandEvent& WXUNUSED(event))
{
  Close(true);
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
```

```
#include "menu.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{
    SubMenu *smenu = new SubMenu(wxT("Submenu"));
    smenu->Show(true);

    return true;
}
```

We created one submenu in a file menu. It is an import submenu, which can be seen in Opera web browser.

```
file->AppendSeparator();
```

A menu separator line is created calling an *AppendSeparator()* method.

```
imp = new wxMenu;
imp->Append(wxID_ANY, wxT("Import newsfeed list..."));
imp->Append(wxID_ANY, wxT("Import bookmarks..."));
imp->Append(wxID_ANY, wxT("Import mail..."));
```

```
file->AppendSubMenu(imp, wxT("I&mport"));
```

A submenu is created like a normal menu. It is appended with a *AppendSubMenu()* method.

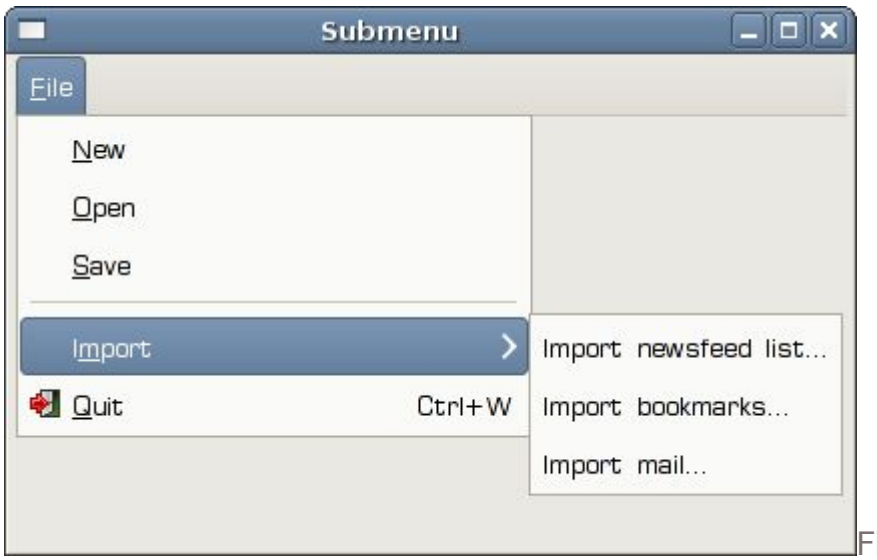


Figure: Submenu

Toolbars

Menus group all commands that we can use in an application. Toolbars provide a quick access to the most frequently used commands.

```
virtual wxToolBar* CreateToolBar(long style = wxNO_BORDER| wxTB_HORIZONTAL,
           wxWindowID id = -1, const wxString& name = "toolBar")
```

To create a toolbar, we call the CreateToolBar() method of the frame widget.

A simple toolbar

Our first example will create a simple toolbar.

toolbar.h

```
#include <wx/wx.h>

class Toolbar : public wxFrame
{
public:
    Toolbar(const wxString& title);

    void OnQuit(wxCommandEvent& event);
```

```
};
```

toolbar.cpp

```
#include "toolbar.h"

Toolbar::Toolbar(const wxString& title)
      : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(280,
180))
{
  wxImage::AddHandler( new wxPNGHandler );

  wxBitmap exit(wxT("exit.png"), wxBITMAP_TYPE_PNG);

  wxToolBar *toolbar = CreateToolBar();  //创建一个工具栏
  toolbar->AddTool(wxID_EXIT, exit, wxT("Exit application"));
  toolbar->Realize();

  Connect(wxID_EXIT, wxEVT_COMMAND_TOOL_CLICKED,
      wxCommandEventHandler(Toolbar::OnQuit));

  Centre();
}

void Toolbar::OnQuit(wxCommandEvent& WXUNUSED(event))
{
  Close(true);
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "toolbar.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{
    Toolbar *toolbar = new Toolbar(wxT("Toolbar"));
    toolbar->Show(true);

    return true;
}
```

In our example, we create a toolbar and one tool button. Clicking on the toolbar button will terminate the application.

```
wxToolBar *toolbar = CreateToolBar();
```

We create a toolbar.

```
toolbar->AddTool(wxID_EXIT, exit, wxT("Exit application"));
```

AddTool 方法的语法是(在文档及头文件中提供的其他重载函数中，也暂没有找到本文第二个参数即图形的方式，但提示中有，运行程序也正确)：

```
wxToolBarToolBase* wxToolBar::AddTool  ( int  toolId,
  const wxString &  label,
  const wxBitmap &  bitmap,
  const wxBitmap &  bmpDisabled,
  wxItemKind  kind = wxITEM_NORMAL,
  const wxString &  shortHelpString = wxEmptyString,
  const wxString &  longHelpString = wxEmptyString,
  wxObject *  clientData = NULL
 )
```

We add a tool to the toolbar.

```
toolbar->Realize();
```

After we have added the tools, we call the *Realize()* method.

Figure: Toolbar

Toolbars

If we want to have more than one toolbar, we must create them in a different way. e.g. other than calling the *CreateToolbar()* method.

toolbars.h

```
#include <wx/wx.h>

class Toolbar : public wxFrame
{
public:
  Toolbar(const wxString& title);

  void OnQuit(wxCommandEvent& event);

  wxToolBar *toolbar1;
  wxToolBar *toolbar2;

};
```

toolbars.cpp

```
#include "toolbars.h"
```

```
Toolbar::Toolbar(const wxString& title)
       : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(280,
180))
{

  wxImage::AddHandler( new wxPNGHandler );

  wxBitmap exit(wxT("exit.png"), wxBITMAP_TYPE_PNG);
  wxBitmap newb(wxT("new.png"), wxBITMAP_TYPE_PNG);
  wxBitmap open(wxT("open.png"), wxBITMAP_TYPE_PNG);
  wxBitmap save(wxT("save.png"), wxBITMAP_TYPE_PNG);

  wxBoxSizer *vbox = new wxBoxSizer(wxVERTICAL);

  toolbar1 = new wxToolBar(this, wxID_ANY);
  toolbar1->AddTool(wxID_ANY, newb, wxT(""));
  toolbar1->AddTool(wxID_ANY, open, wxT(""));
  toolbar1->AddTool(wxID_ANY, save, wxT(""));
  toolbar1->Realize();

  toolbar2 = new wxToolBar(this, wxID_ANY);
  toolbar2->AddTool(wxID_EXIT, exit, wxT("Exit application"));
  toolbar2->Realize();

  vbox->Add(toolbar1, 0, wxEXPAND);
  vbox->Add(toolbar2, 0, wxEXPAND);

  SetSizer(vbox);

  Connect(wxID_EXIT, wxEVT_COMMAND_TOOL_CLICKED,
      wxCommandEventHandler(Toolbar::OnQuit));

  Centre();
}

void Toolbar::OnQuit(wxCommandEvent& WXUNUSED(event))
{
  Close(true);
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "toolbars.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{
    Toolbar *toolbar = new Toolbar(wxT("Toolbar"));
    toolbar->Show(true);

    return true;
}
```

In our example, we create two horizontal toolbars. We place them in a vertical box sizer.

```
toolbar1 = new wxToolBar(this, wxID_ANY);
...
toolbar2 = new wxToolBar(this, wxID_ANY);
```

Here we create two toolbars.

```
vbox->Add(toolbar1, 0, wxEXPAND);
vbox->Add(toolbar2, 0, wxEXPAND);
```

And here we add them to the vertical box sizer.

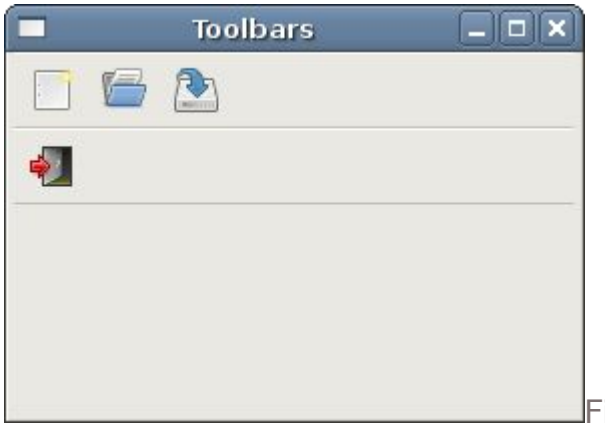


Figure: Toolbars

In this part of the wxWidgets tutorial, we have covered menus and toolbars.

Layout management in wxWidgets

A typical application consists of various widgets. Those widgets are placed inside container widgets. A programmer must manage the layout of the application. This is not an easy task. In wxWidgets we have two options.

- absolute positioning
- sizers

Absolute Positioning

The programmer specifies the position and the size of each widget in pixels. When you use absolute positioning, you have to understand several things.

- the size and the position of a widget do not change, if you resize a window
- applications look different (crappy) on various platforms
- changing fonts in your application might spoil the layout
- if you decide to change your layout, you must completely redo your layout, which is tedious and time consuming

There might be situations, where we can possibly use absolute positioning. For example, my tutorials. I do not want to make the examples too difficult, so I often use absolute positioning to explain a topic. But mostly, in real world programs, programmers use sizers.

In our example we have a simple skeleton of a text editor. If we resize the window, the size of out wxTextCtrl does not change as we would expect.

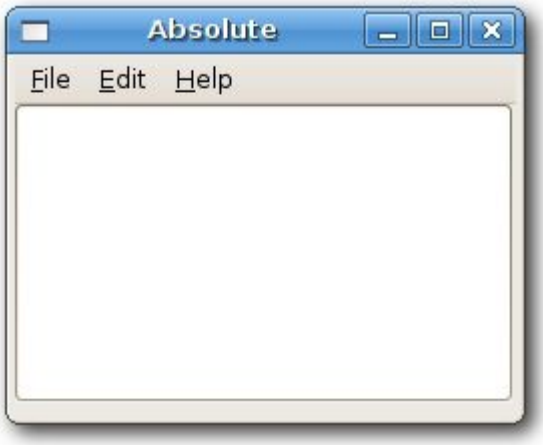


Figure: before resizement

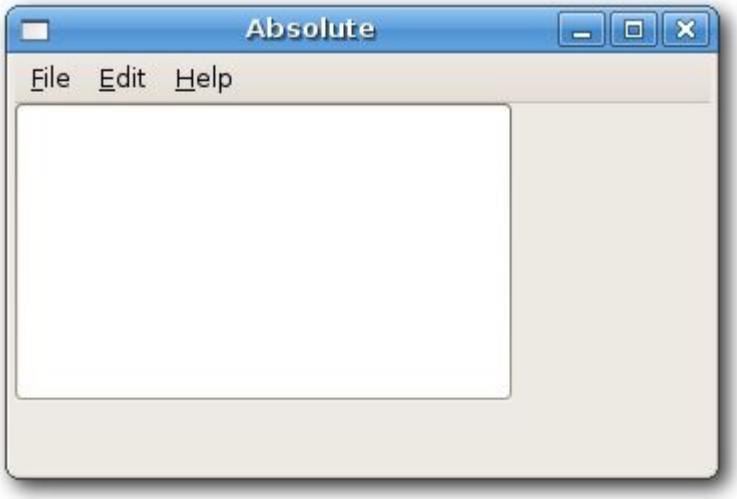


Figure: after resizement

absolute.h

```
#include <wx/wx.h>

class Absolute : public wxFrame
{
public:
  Absolute(const wxString& title);

  wxMenuBar *menubar;
  wxMenu *file;
  wxMenu *edit;
```

```
  wxMenu *help;
  wxTextCtrl *textctrl;
};
```

absolute.cpp

```
#include "absolute.h"

Absolute::Absolute(const wxString& title)
       : wxFrame(NULL, -1, title, wxPoint(-1, -1), wxSize(250, 180))
{
 wxPanel *panel = new wxPanel(this, -1);  //-1处该是 Panel 的 id

 menubar = new wxMenuBar;
 file = new wxMenu;
 edit = new wxMenu;
 help = new wxMenu;

 menubar->Append(file, wxT("&File"));
 menubar->Append(edit, wxT("&Edit"));
 menubar->Append(help, wxT("&Help"));
 SetMenuBar(menubar);

 textctrl = new wxTextCtrl(panel, -1, wxT(""), wxPoint(-1, -1),
     wxSize(250, 150));

 Centre();
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "absolute.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{

    Absolute *absolute = new Absolute(wxT("Absolute"));
    absolute->Show(true);

    return true;
}
```

This is an example, where we use absolute positioning. We position a *wxTextCtrl* widget on a panel widget.

```
 textctrl = new wxTextCtrl(panel, -1, wxT(""), wxPoint(-1, -1),
    wxSize(250, 150));
```

We do the absolute positioning in the constructor of the *wxTextCtrl* widget. In our case, we provide the default position for the widget. The width is 250px and the height 150px.

Using sizers

Sizers in wxWidgets do address all those issues, we mentioned by absolute positioning. We can choose among these sizers.

- wxBoxSizer
- wxStaticBoxSizer
- wxGridSizer
- wxFlexGridSizer
- wxGridBagSizer

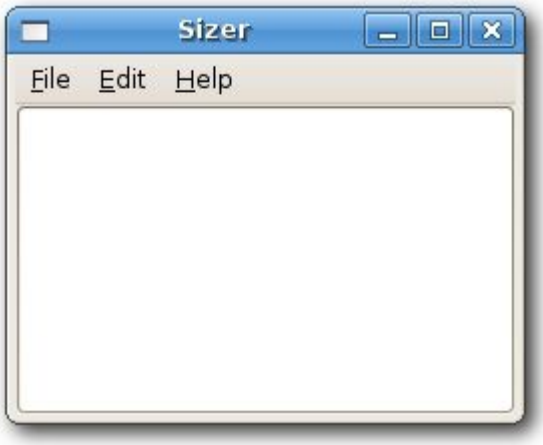


Figure: before resizement

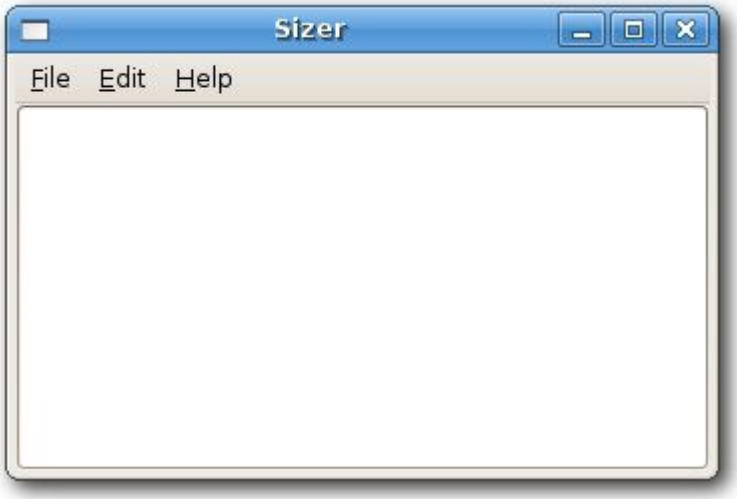


Figure: after resizement

sizer.h

```
#include <wx/wx.h>

class Sizer : public wxFrame
{
public:
  Sizer(const wxString& title);

  wxMenuBar *menubar;
  wxMenu *file;
  wxMenu *edit;
  wxMenu *help;
  wxTextCtrl *textctrl;
```

```
};
```

sizer.cpp

```
#include "sizer.h"

Sizer::Sizer(const wxString& title)
       : wxFrame(NULL, -1, title, wxPoint(-1, -1), wxSize(250, 180))
{
//这儿同 class Absolute 有了区别，Absolute 在这里新建了一个 panel
 menubar = new wxMenuBar;
 file = new wxMenu;
 edit = new wxMenu;
 help = new wxMenu;

 menubar->Append(file, wxT("&File"));
 menubar->Append(edit, wxT("&Edit"));
 menubar->Append(help, wxT("&Help"));
 SetMenuBar(menubar);

 textctrl = new wxTextCtrl(this, -1, wxT(""), wxPoint(-1, -1),
     wxSize(250, 150));  //本例将 textctrl 置入 this，而 Absolute 放入了
panel

 Centre();
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
```

```
#include "sizer.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{

    Sizer *sizer = new Sizer(wxT("Sizer"));
    sizer->Show(true);

    return true;
}
```

Ok, so you are saying that you don't see any sizers in the example? Well, the code example was a bit tricky(狡猾). Actually, we placed the wxTextCtrl inside the wxFrame widget. The wxFrame widget has a special built-in sizer. We can put only one widget inside the wxFrame container. The child widget occupies all the space, which is not given to the borders, menu, toolbar and the statusbar.

wxBoxSizer

This sizer enables us to put several widgets into a row or a column. We can put another sizer into an existing sizer. This way we can create very complex layouts.

```
wxBoxSizer(int orient)
wxSizerItem* Add(wxWindow* window, int proportion = 0, int flag = 0, int
border = 0)
```

The orientation can be wxVERTICAL or wxHORIZONTAL. Adding widgets into the wxBoxSizer is done via the Add() method. In order to understand it, we need to look at its parameters.

The proportion parameter defines the ratio of how will the widgets change in the defined orientation. Let's assume we have tree buttons with the proportions 0, 1, and 2. They are added into a horizontal wxBoxSizer. Button

with proportion 0 will not change at all. Button with proportion 2 will change twice more than the one with proportion 1 in the horizontal dimension.

With the flag parameter you can further configure the behaviour of the widgets within a wxBoxSizer. We can control the border between the widgets. We add some space between widgets in pixels. In order to apply border we need to define sides, where the border will be used. We can combine them with the | operator. e.g wxLEFT | wxBOTTOM. We can choose between these flags:

- wxLEFT
- wxRIGHT
- wxBOTTOM
- wxTOP
- wxALL

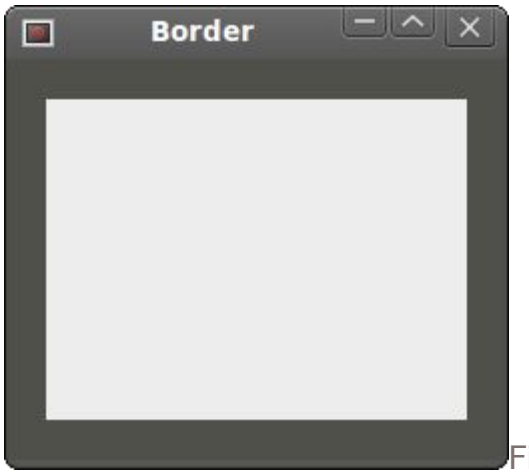


Figure: border around a panel

border.h

```
#include <wx/wx.h>

class Border : public wxFrame
{
public:
  Border(const wxString& title);
};
```

border.cpp

```
#include "border.h"

Border::Border(const wxString& title)
       : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(250,
200))
{
  wxColour col1, col2;
  col1.Set(wxT("#4f5049"));
  col2.Set(wxT("#ededed"));

  wxPanel *panel = new wxPanel(this, -1);
  panel->SetBackgroundColour(col1);
  wxBoxSizer *vbox = new wxBoxSizer(wxVERTICAL);

  wxPanel *midPan = new wxPanel(panel, wxID_ANY);
  midPan->SetBackgroundColour(col2);

  vbox->Add(midPan, 1, wxEXPAND | wxALL, 20);
  panel->SetSizer(vbox);

  Centre();
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "border.h"

IMPLEMENT_APP(MyApp)
```

```
bool MyApp::OnInit()
{

    Border *border = new Border(wxT("Border"));
    border->Show(true);

    return true;
}
```

In this example, we create two panels. The second panel has some space around itself.

```
 vbox->Add(midPan, 1, wxEXPAND | wxALL, 20);
```

We have placed a 20 px border around a midPan panel. wxALL applies the border size to all four sides. If we use wxEXPAND flag, the widget will use all the space that has been allotted to it.

Lastly, we can also define the alignment of our widgets. We do it with the following flags :

- wxALIGN_LEFT
- wxALIGN_RIGHT
- wxALIGN_TOP
- wxALIGN_BOTTOM
- wxALIGN_CENTER_VERTICAL
- wxALIGN_CENTER_HORIZONTAL
- wxALIGN_CENTER

Say we wanted to place two buttons into the right bottom of the window.

align.h

```
#include <wx/wx.h>

class Align : public wxFrame
{
public:
```

```
    Align(const wxString& title);

};
```

align.cpp

```
#include "align.h"

Align::Align(const wxString& title)
       : wxFrame(NULL, -1, title, wxPoint(-1, -1), wxSize(300, 200))
{

  wxPanel *panel = new wxPanel(this, -1);

  wxBoxSizer *vbox = new wxBoxSizer(wxVERTICAL);
  wxBoxSizer *hbox1 = new wxBoxSizer(wxHORIZONTAL);
  wxBoxSizer *hbox2 = new wxBoxSizer(wxHORIZONTAL);

  wxButton *ok = new wxButton(panel, -1, wxT("Ok"));
  wxButton *cancel = new wxButton(panel, -1, wxT("Cancel"));

  hbox1->Add(new wxPanel(panel, -1));
  vbox->Add(hbox1, 1, wxEXPAND);


  hbox2->Add(ok);
  hbox2->Add(cancel);

  vbox->Add(hbox2, 0, wxALIGN_RIGHT | wxRIGHT | wxBOTTOM, 10);
  panel->SetSizer(vbox);

  Centre();
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
```

```
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "align.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{

    Align *align = new Align(wxT("Align"));
    align->Show(true);

    return true;
}
```

We create three sizers. One vertical sizer and two horizontal sizers. We put those two horizontal sizers into the vertical one.

```
hbox1->Add(new wxPanel(panel, -1));
vbox->Add(hbox1, 1, wxEXPAND);
```

We put a *wxPanel* into the first horizontal sizer. We set the proportion to *1* and set a *wxEXPAND*flag. This way the sizer will occupy all the space except the *hbox2*.

```
vbox->Add(hbox2, 0, wxALIGN_RIGHT | wxRIGHT | wxBOTTOM, 10);
```

We have placed the buttons into the *hbox2* sizer. The hbox2 is right aligned and we also put some space to the bottom and to the right of the buttons.

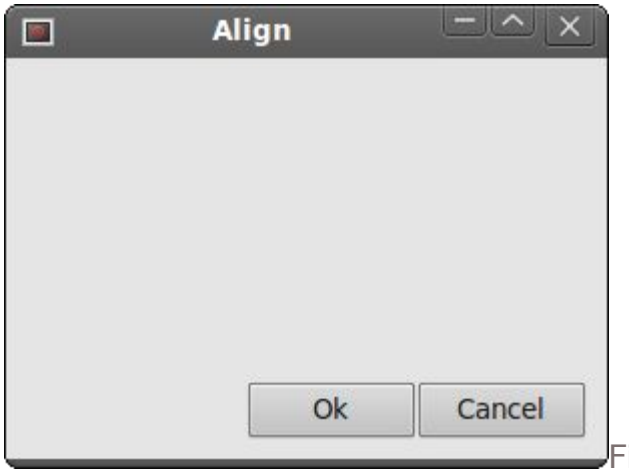


Figure: aligning buttons

Go To Class

In the following example we introduce several important ideas.

gotoclass.h

```
#include <wx/wx.h>

class GotoClass : public wxFrame
{
public:
  GotoClass(const wxString& title);
};
```

gotoclass.cpp

```
#include "gotoclass.h"

GotoClass::GotoClass(const wxString& title)
       : wxFrame(NULL, -1, title, wxPoint(-1, -1), wxSize(450, 400))
{
  wxPanel *panel = new wxPanel(this, -1);

  wxBoxSizer *vbox = new wxBoxSizer(wxVERTICAL);

  wxBoxSizer *hbox1 = new wxBoxSizer(wxHORIZONTAL);
  wxStaticText *st1 =  new wxStaticText(panel, wxID_ANY,
      wxT("Class Name"));
```

```
hbox1->Add(st1, 0, wxRIGHT, 8);
wxTextCtrl *tc = new wxTextCtrl(panel, wxID_ANY);
hbox1->Add(tc, 1);
vbox->Add(hbox1, 0, wxEXPAND | wxLEFT | wxRIGHT | wxTOP, 10);

vbox->Add(-1, 10);

wxBoxSizer *hbox2 = new wxBoxSizer(wxHORIZONTAL);
wxStaticText *st2 = new wxStaticText(panel, wxID_ANY,
    wxT("Matching Classes"));

hbox2->Add(st2, 0);
vbox->Add(hbox2, 0, wxLEFT | wxTOP, 10);

vbox->Add(-1, 10);

wxBoxSizer *hbox3 = new wxBoxSizer(wxHORIZONTAL);
wxTextCtrl *tc2 = new wxTextCtrl(panel, wxID_ANY, wxT(""),
    wxPoint(-1, -1), wxSize(-1, -1), wxTE_MULTILINE);

hbox3->Add(tc2, 1, wxEXPAND);
vbox->Add(hbox3, 1, wxLEFT | wxRIGHT | wxEXPAND, 10);

vbox->Add(-1, 25);

wxBoxSizer *hbox4 = new wxBoxSizer(wxHORIZONTAL);
wxCheckBox *cb1 = new wxCheckBox(panel, wxID_ANY,
    wxT("Case Sensitive"));

hbox4->Add(cb1);
wxCheckBox *cb2 = new wxCheckBox(panel, wxID_ANY,
    wxT("Nested Classes"));

hbox4->Add(cb2, 0, wxLEFT, 10);
wxCheckBox *cb3 = new wxCheckBox(panel, wxID_ANY,
    wxT("Non-Project Classes"));

hbox4->Add(cb3, 0, wxLEFT, 10);
vbox->Add(hbox4, 0, wxLEFT, 10);

vbox->Add(-1, 25);

wxBoxSizer *hbox5 = new wxBoxSizer(wxHORIZONTAL);
```

```
  wxButton *btn1 = new wxButton(panel, wxID_ANY, wxT("Ok"));
  hbox5->Add(btn1, 0);
  wxButton *btn2 = new wxButton(panel, wxID_ANY, wxT("Close"));
  hbox5->Add(btn2, 0, wxLEFT | wxBOTTOM , 5);
  vbox->Add(hbox5, 0, wxALIGN_RIGHT | wxRIGHT, 10);

  panel->SetSizer(vbox);

  Centre();
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "gotoclass.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{
 GotoClass *gotoclass = new GotoClass(wxT("GotoClass"));
 gotoclass->Show(true);

 return true;
}
```

This is a complex example using *wxBoxSizer*. The layout is straitforward. We create one vertical sizer. We put then five horizontal sizers into it.

```
vbox->Add(hbox3, 1, wxLEFT | wxRIGHT | wxEXPAND, 10);
vbox->Add(-1, 25);
```

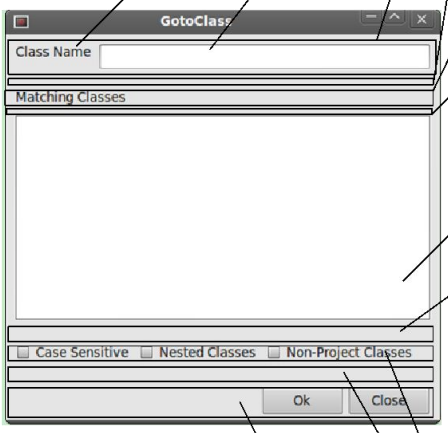
We already know that we can control the distance among widgets by combining the flag parameter with the border parameter. But there is one real constraint. In the Add() method we can specify only one border for all given sides. In our example, we give 10px to the right and to the left. But we cannot give 25 px to the bottom. What we can do is to give 10px to the bottom, or 0px. If we omit wxBOTTOM. So if we need different values, we can add some extra space. With the Add() method, we can insert widgets and space as well.

```
vbox->Add(hbox5, 0, wxALIGN_RIGHT | wxRIGHT, 10);
```

We place the two buttons on the right side of the window. How do we do it? Three things are important to achieve this. The proportion, the align flag and the wxEXPAND flag. The proportion must be zero. The buttons should not change their size, when we resize our window. We must not specify wxEXPAND flag. The buttons occopy only the area that has been alotted to it. And finally, we must specify the wxALIGN_RIGHT flag. The horizontal sizer spreads from the left side of the window to the right side. So if we specify wxALIGN_RIGHT flag, the buttons are placed to the right side. Exactly, as we wanted.

Figure: GotoClass

下图将代码与运行结果联系起来，便于体会。几处 vbox->Add(-1, 25)，是用编号为-1 的 Panel，在垂直方向上分隔，以使好看一些。




```
GotoClass::GotoClass(const wxString& title)
      : wxFrame(NULL, -1, title, wxPoint(-1, -1), wxSize(450, 400))
{
  wxPanel *panel = new wxPanel(this, -1);

  wxBoxSizer *vbox = new wxBoxSizer(wxVERTICAL);

  wxBoxSizer *hbox1 = new wxBoxSizer(wxHORIZONTAL);
  wxStaticText *st1 =  new wxStaticText(panel, wxID_ANY,
     wxT("Class Name"));

  hbox1->Add(st1, 0, wxRIGHT, 8);
  wxTextCtrl *tc = new wxTextCtrl(panel, wxID_ANY);
  hbox1->Add(tc, 1);
  vbox->Add(hbox1, 0, wxEXPAND | wxLEFT | wxRIGHT | wxTOP, 10);

  vbox->Add(-1, 10);

  wxBoxSizer *hbox2 = new wxBoxSizer(wxHORIZONTAL);
  wxStaticText *st2 = new wxStaticText(panel, wxID_ANY,
     wxT("Matching Classes"));

  hbox2->Add(st2, 0);
  vbox->Add(hbox2, 0, wxLEFT | wxTOP, 10);

  vbox->Add(-1, 10);

  wxBoxSizer *hbox3 = new wxBoxSizer(wxHORIZONTAL);
  wxTextCtrl *tc2 = new wxTextCtrl(panel, wxID_ANY, wxT(""),
     wxPoint(-1, -1), wxSize(-1, -1), wxTE_MULTILINE);

  hbox3->Add(tc2, 1, wxEXPAND);
  vbox->Add(hbox3, 1, wxLEFT | wxRIGHT | wxEXPAND, 10);

  vbox->Add(-1, 25);

  wxBoxSizer *hbox4 = new wxBoxSizer(wxHORIZONTAL);
  wxCheckBox *cb1 = new wxCheckBox(panel, wxID_ANY,
     wxT("Case Sensitive"));

  hbox4->Add(cb1);
  wxCheckBox *cb2 = new wxCheckBox(panel, wxID_ANY,
     wxT("Nested Classes"));

  hbox4->Add(cb2, 0, wxLEFT, 10);
  wxCheckBox *cb3 = new wxCheckBox(panel, wxID_ANY,
     wxT("Non-Project Classes"));

  hbox4->Add(cb3, 0, wxLEFT, 10);
  vbox->Add(hbox4, 0, wxLEFT, 10);

  vbox->Add(-1, 25);

  wxBoxSizer *hbox5 = new wxBoxSizer(wxHORIZONTAL);
  wxButton *btn1 = new wxButton(panel, wxID_ANY, wxT("Ok"));
  hbox5->Add(btn1, 0);
  wxButton *btn2 = new wxButton(panel, wxID_ANY, wxT("Close"));
  hbox5->Add(btn2, 0, wxLEFT | wxBOTTOM , 5);
  vbox->Add(hbox5, 0, wxALIGN_RIGHT | wxRIGHT, 10);

  panel->SetSizer(vbox);

  Centre();
}
```

wxGridSizer

wxGridSizer lays out widgets in two dimensional table. Each cell within the table has the same size.

```
wxGridSizer(int rows, int cols, int vgap, int hgap)
```

In the constructor we specify the number of rows and columns in the table. And the vertical and horizontal space between our cells.

In our example we create a skeleton of a calculator. It is a perfect example for a wxGridSizer.

gridsizer.h

```
#include <wx/wx.h>

class GridSizer : public wxFrame
{
public:
  GridSizer(const wxString& title);

  wxMenuBar *menubar;
  wxMenu *file;

  wxBoxSizer *sizer;
  wxGridSizer *gs;
  wxTextCtrl *display;
};
```

gridsizer.cpp

```
#include "gridsizer.h"

GridSizer::GridSizer(const wxString& title)
       : wxFrame(NULL, -1, title, wxPoint(-1, -1), wxSize(270, 220))
{
  menubar = new wxMenuBar;
  file = new wxMenu;
```

```
  SetMenuBar(menubar);

  sizer = new wxBoxSizer(wxVERTICAL);

  display = new wxTextCtrl(this, -1, wxT(""), wxPoint(-1, -1),
     wxSize(-1, -1), wxTE_RIGHT);
  sizer->Add(display, 0, wxEXPAND | wxTOP | wxBOTTOM, 4);

  gs = new wxGridSizer(4, 4, 3, 3);  //改为(5, 4, 3, 3)
  gs->Add(new wxButton(this, -1, wxT("Cls")), 0, wxEXPAND);
  gs->Add(new wxButton(this, -1, wxT("Bck")), 0, wxEXPAND);
  gs->Add(new wxStaticText(this, -1, wxT("")), 0, wxEXPAND);
  gs->Add(new wxButton(this, -1, wxT("Close")), 0, wxEXPAND);
  gs->Add(new wxButton(this, -1, wxT("7")), 0, wxEXPAND);
  gs->Add(new wxButton(this, -1, wxT("8")), 0, wxEXPAND);
  gs->Add(new wxButton(this, -1, wxT("9")), 0, wxEXPAND);
  gs->Add(new wxButton(this, -1, wxT("/")), 0, wxEXPAND);
  gs->Add(new wxButton(this, -1, wxT("4")), 0, wxEXPAND);
  gs->Add(new wxButton(this, -1, wxT("5")), 0, wxEXPAND);
  gs->Add(new wxButton(this, -1, wxT("6")), 0, wxEXPAND);
  gs->Add(new wxButton(this, -1, wxT("*")), 0, wxEXPAND);
  gs->Add(new wxButton(this, -1, wxT("1")), 0, wxEXPAND);
  gs->Add(new wxButton(this, -1, wxT("2")), 0, wxEXPAND);
  gs->Add(new wxButton(this, -1, wxT("3")), 0, wxEXPAND);
  gs->Add(new wxButton(this, -1, wxT("-")), 0, wxEXPAND);
  gs->Add(new wxButton(this, -1, wxT("0")), 0, wxEXPAND);
  gs->Add(new wxButton(this, -1, wxT(".")), 0, wxEXPAND);
  gs->Add(new wxButton(this, -1, wxT("=")), 0, wxEXPAND);
  gs->Add(new wxButton(this, -1, wxT("+")), 0, wxEXPAND);

  sizer->Add(gs, 1, wxEXPAND);
  SetSizer(sizer);
  SetMinSize(wxSize(270, 220));

  Centre();
}
```

main.h

```
#include <wx/wx.h>
```

```
class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "gridsizer.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{

    GridSizer *gs = new GridSizer(wxT("GridSizer"));
    gs->Show(true);

    return true;
}
```

In our example, we set a vertical sizer for a wxFrame. We put a static text and a grid sizer into the vertical sizer.

Notice how we managed to put a space between the Bck and the Close buttons. We simply put an empty wxStaticText there. Such tricks are quite common.

```
gs->Add(new wxButton(this, -1, wxT("Cls")), 0, wxEXPAND);
```

We call the *Add()* method multiple times. Widgets are placed inside the table in the order, they are added. The first row is filled first, then the second row etc.

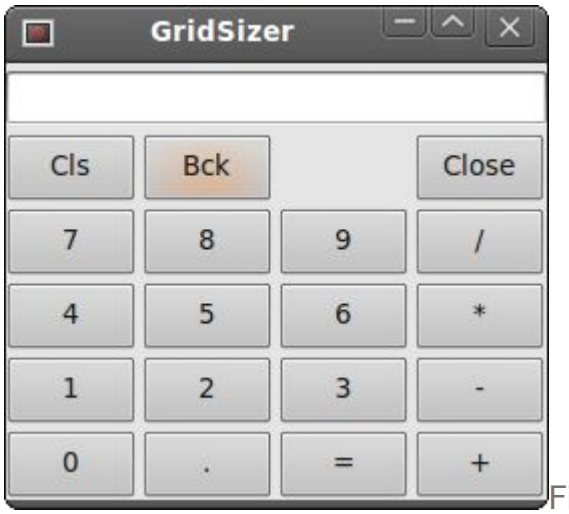


Figure: GridSizer

wxFlexGridSizer

This sizer is similar to *wxGridSizer*. It does also lay out it's widgets in a two dimensional table. It adds some flexibility to it. *wxGridSizer* cells are of the same size. All cells in *wxFlexGridSizer* have the same height in a row. All cells have the same width in a column. But all rows and columns are not necessarily the same height or width.

```
wxFlexGridSizer(int rows, int cols, int vgap, int hgap)
```

rows and *cols* specify the number of rows and columns in a sizer. *vgap* and *hgap* add some space between widgets in both directions.

Many times developers have to develop dialogs for data input and modification. I find *wxFlexGridSizer* suitable for such a task. A developer can easily set up a dialog window with this sizer. It is also possible to accomplish this with *wxGridSizer*, but it would not look nice, because of the constraint that each cell has the same size.

flexgridsizer.h

```
#include <wx/wx.h>

class FlexGridSizer : public wxFrame
{
```

```
public:
  FlexGridSizer(const wxString& title);
};
```

flexgridsizer.cpp

```
#include "flexgridsizer.h"

FlexGridSizer::FlexGridSizer(const wxString& title)
       : wxFrame(NULL, -1, title, wxPoint(-1, -1), wxSize(270, 220))
{
  wxPanel *panel = new wxPanel(this, -1);

  wxBoxSizer *hbox = new wxBoxSizer(wxHORIZONTAL);

  wxFlexGridSizer *fgs = new wxFlexGridSizer(3, 2, 9, 25);//3行2列，垂直和水平方向隔开9和25像素

  wxStaticText *thetitle = new wxStaticText(panel, -1, wxT("Title"));
  wxStaticText *author = new wxStaticText(panel, -1, wxT("Author"));
  wxStaticText *review = new wxStaticText(panel, -1, wxT("Review"));

  wxTextCtrl *tc1 = new wxTextCtrl(panel, -1);
  wxTextCtrl *tc2 = new wxTextCtrl(panel, -1);
  wxTextCtrl *tc3 = new wxTextCtrl(panel, -1, wxT(""),
      wxPoint(-1, -1), wxSize(-1, -1), wxTE_MULTILINE);

  fgs->Add(thetitle);
  fgs->Add(tc1, 1, wxEXPAND);
  fgs->Add(author);
  fgs->Add(tc2, 1, wxEXPAND);
  fgs->Add(review, 1, wxEXPAND);
  fgs->Add(tc3, 1, wxEXPAND);

  fgs->AddGrowableRow(2, 1); //编号为2的那一行(从0数)，缩放比例为1
  fgs->AddGrowableCol(1, 1); //编号为1的那一列(从0数)，缩放比例为1

  hbox->Add(fgs, 1, wxALL | wxEXPAND, 15);
  panel->SetSizer(hbox);
  Centre();
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "flexgridsizer.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{

    FlexGridSizer *fgs = new FlexGridSizer(wxT("FlexGridSizer"));
    fgs->Show(true);

    return true;
}
```

In our example we create a simple dialog. It could be used to insert data into the database.

```
wxBoxSizer *hbox = new wxBoxSizer(wxHORIZONTAL);
...
hbox->Add(fgs, 1, wxALL | wxEXPAND, 15);
```

We create a horizontal box sizer in order to put some space (15px) around the table of widgets.

```
fgs->Add(thetitle);
```

We add widgets to the sizer exactly as with the gridsizer.

```
fgs->AddGrowableRow(2, 1);
fgs->AddGrowableCol(1, 1);
```

We make the third row and the second column growable. This way we make the text controls grow, when the window is resized. The first two text controls will grow in horizontal direction, the third one will grow in both direction. We must not forget to make the widgets expandable (wxEXPAND) in order to make it really work.

Figure: FlexGridSizer

This part of the wxWidgets tutorial was dedicated to layout management.

Events

Events are integral part of every GUI application. All GUI applications are event-driven. An application reacts to different event types which are generated during its lifetime. Events are generated mainly by the user of an application. But they can be generated by other means as well. e.g. internet connection, window manager, timer. When the application starts a main loop is created. The application sits in the main loop and waits for the events to be generated. The main loop quits, when we exit the application.

（理解事件很重要，译一段出来）事件是每一个 GUI 应用程序的集成部分。所有的 GUI 应用程序都是事件驱动的。一个应用程序，在其生存期间，总是在和产生的各种不同类型的事件进行交互。事件主要由用户在操作中产生，不过也有其他途径产生的事件，例如，互联网连接、窗口管理、定时器等。当一个应用程序开始时，一个主要的循环也就创建好了。应用程序就在这个循环中，等待事件的发生。当应用程序结束，循环也就退出了。

Definitions

Event is a piece of application-level information from the underlying framework, typically the GUI toolkit. **Event loop** is a programming construct that waits for and dispatches（分派） events or messages in a program. The event loop repeatedly looks for events to process them. A**dispatcher** is a process which maps events to **event handlers**. Event handlers are methods that react to events.

Event object is an object associated with the event. It is usually a window. **Event type** is a unique event, that has been generated.

A simple event example

The traditional way to work with events in wxWidgets is to use **static event tables.** This was influenced by the MFC. A more flexible and modern way is to use the **Connect()** method. Because this way is superior to event tables, I use it throughout the wxWidgets tutorial.

Event table

In the next example, we show an example, where we use event tables.

button.h

```
#include <wx/wx.h>

class MyButton : public wxFrame
{
public:
    MyButton(const wxString& title);

    void OnQuit(wxCommandEvent& event);

private:
    DECLARE_EVENT_TABLE()
};
```

button.cpp

```
#include "button.h"

MyButton::MyButton(const wxString& title)
       : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(270,
150))
{

  wxPanel *panel = new wxPanel(this, wxID_ANY);
  wxButton *button = new wxButton(panel, wxID_EXIT,
      wxT("Quit"), wxPoint(20, 20));

  Centre();
}

void MyButton::OnQuit(wxCommandEvent& WXUNUSED(event))
{
    Close(true);
}

BEGIN_EVENT_TABLE(MyButton, wxFrame)
```

```
    EVT_BUTTON(wxID_EXIT,  MyButton::OnQuit)
END_EVENT_TABLE()
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "button.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{

    MyButton *button = new MyButton(wxT("Button"));
    button->Show(true);

    return true;
}
```

In our example we create a simple button. By clicking on the button, we close the application.

```
private:
    DECLARE_EVENT_TABLE()
```

In the header file, we declare an event table with the DECLARE_EVENT_TABLE() macro.

```
BEGIN_EVENT_TABLE(MyButton, wxFrame)
    EVT_BUTTON(wxID_EXIT,  MyButton::OnQuit)
END_EVENT_TABLE()
```

We implement an event table by mapping each event to the appropriate member function.

Example using Connect()

We will talk about a move event. A move event holds information about move change events. A move event is generated, when we move a window to a new position. The class that represents the move event is **wxMoveEvent**. The **wxEVT_MOVE** is an event type.

move.h

```
#include <wx/wx.h>

class Move : public wxFrame
{
public:
  Move(const wxString& title);

  void OnMove(wxMoveEvent & event);

  wxStaticText *st1;
  wxStaticText *st2;
};
```

move.cpp

```
#include "move.h"

Move::Move(const wxString& title)
       : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(250,
130))
{
  wxPanel *panel = new wxPanel(this, -1);
```

```
  //静态文本中的文字暂为空，当窗口移动后由事件修改
  st1 = new wxStaticText(panel, -1, wxT(""), wxPoint(10, 10));
  st2 = new wxStaticText(panel, -1, wxT(""), wxPoint(10, 30));

  Connect(wxEVT_MOVE, wxMoveEventHandler(Move::OnMove));

  Centre();
}

void Move::OnMove(wxMoveEvent& event)
{
  wxPoint size = event.GetPosition();
  st1->SetLabel(wxString::Format(wxT("x: %d"), size.x ));
  st2->SetLabel(wxString::Format(wxT("y: %d"), size.y ));
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "move.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{
    Move *move = new Move(wxT("Move event"));
    move->Show(true);

    return true;
}
```

The example displays the current position of the window.

```
Connect(wxEVT_MOVE, wxMoveEventHandler(Move::OnMove));
```

Here we connect a *wxEVT_MOVE* event type with the *OnMove()* method.

```
wxPoint size = event.GetPosition();
```

The event parameter in the *OnMove()* method is an object specific to a particular event. In our case it is the instance of a *wxMoveEvent* class. This object holds information about the event. We can find out the current position by calling the *GetPosition()* method of the event.

Figure: Move event

Event propagation

There are two types of events. Basic events and command events. They differ in propagation. Event propagation is travelling of events from child widgets to parent widgets and grand parent widgets etc. Basic events do not propagate. Command events do propagate. For example *wxCloseEvent* is a basic event. It does not make sense for this event to propagate to parent widgets.

By default, the event that is catched in a event handler stops propagating. To continue propagation, we must call the *Skip()* method.

propagate.h

```
#include <wx/wx.h>
```

```
class Propagate : public wxFrame
{
public:
  Propagate(const wxString& title);

  void OnClick(wxCommandEvent& event);
};

class MyPanel : public wxPanel
{
public:
  MyPanel(wxFrame *frame, int id);

  void OnClick(wxCommandEvent& event);
};

class MyButton : wxButton
{
public:
  MyButton(MyPanel *panel, int id, const wxString &label);

  void OnClick(wxCommandEvent& event);
};
```

propagate.cpp

```
#include <iostream>
#include "propagate.h"

const int ID_BUTTON = 1;

Propagate::Propagate(const wxString& title)
    : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(250,
130))
{
  MyPanel *panel = new MyPanel(this, -1);  //panel 在 frame 中

  new MyButton(panel, ID_BUTTON, wxT("Ok")); //button 在 panel 中

  Connect(ID_BUTTON, wxEVT_COMMAND_BUTTON_CLICKED,
      wxCommandEventHandler(Propagate::OnClick));
```

```
  Centre();
}

void Propagate::OnClick(wxCommandEvent& event)
{
  std::cout << "event reached frame class" << std::endl;
  event.Skip();
}

MyPanel::MyPanel(wxFrame *frame, int id)
    : wxPanel(frame, id)
{
  Connect(ID_BUTTON, wxEVT_COMMAND_BUTTON_CLICKED,
      wxCommandEventHandler(MyPanel::OnClick));
}

void MyPanel::OnClick(wxCommandEvent& event)
{
  std::cout << "event reached panel class" << std::endl;
  event.Skip();
}

MyButton::MyButton(MyPanel *mypanel, int id, const wxString& label)
    : wxButton(mypanel, id, label, wxPoint(15, 15))
{
  Connect(ID_BUTTON, wxEVT_COMMAND_BUTTON_CLICKED,
      wxCommandEventHandler(MyButton::OnClick));
}

void MyButton::OnClick(wxCommandEvent& event)
{
  std::cout << "event reached button class" << std::endl;
  event.Skip();
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
```

```
  public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "propagate.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{

    Propagate *prop = new Propagate(wxT("Propagate"));
    prop->Show(true);

    return true;
}
```

In our example, we have a button on a panel. The panel is placed in a frame widget. We define a handler for all widgets.

```
event reached button class
event reached panel class
event reached frame class
```

We get this, when we click on the button. The event travels from the button to panel and to frame.

Try to omit some Skip() methods and see, what hapens.

Vetoing events

Sometimes we need to stop processing an event. To do this, we call the method Veto().

veto.h

```
#include <wx/wx.h>

class Veto : public wxFrame
{
public:
  Veto(const wxString& title);

  void OnClose(wxCloseEvent& event);
};
```

veto.cpp

```
#include "veto.h"

Veto::Veto(const wxString& title)
    : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(250,
130))
{

  Connect(wxEVT_CLOSE_WINDOW, wxCloseEventHandler(Veto::OnClose));
  Centre();
}

void Veto::OnClose(wxCloseEvent& event)
{
  wxMessageDialog *dial = new wxMessageDialog(NULL,
      wxT("Are you sure to quit?"), wxT("Question"),
      wxYES_NO | wxNO_DEFAULT | wxICON_QUESTION);

  int ret = dial->ShowModal();
  dial->Destroy();

  if (ret == wxID_YES) {
      Destroy();
  } else {
      event.Veto();
  }
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "veto.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{

    Veto *veto = new Veto(wxT("Veto"));
    veto->Show(true);

    return true;
}
```

In our example, we process a *wxCloseEvent*. This event is called, when we click the X button on the titlebar, press Alt + F4 or select close from the system menu. In many applications, we want to prevent from accidentally closing the window, if we made some changes. To do this, we must connect the wxEVT_CLOSE_WINDOW event type.

```
wxMessageDialog *dial = new wxMessageDialog(NULL,
   wxT("Are you sure to quit?"), wxT("Question"),
   wxYES_NO | wxNO_DEFAULT | wxICON_QUESTION);
```

During the close event, we show a message dialog.

```
if (ret == wxID_YES) {
    Destroy();
} else {
```

```
    event.Veto();
}
```

Depending on the return value, we destroy the window, or veto the event. Notice that to close the window, we must call the *Destroy()* method. By calling the *Close()* method, we would end up in an endless cycle.

Window identifiers

Window identifiers are integers that uniquely determine the window identity in the event system. There are three ways to create window id's.

- let the system automatically create an id
- use standard identifiers
- create your own id

Each widget has an id parameter. This is a unique number in the event system. If we work with multiple widgets, we must differantiate among them.

```
wxButton(parent, -1)
wxButton(parent, wxID_ANY)
```

If we provide -1 or wxID_ANY for the id parameter, we let the wxWidgets automatically create an id for us. The automatically created id's are always negative, whereas user specified id's must always be positive. We usually use this option when we do not need to change the widget state. For example a static text, that will never be changed during the life of the application. We can still get the id, if we want. There is a method *GetId()*, which will determine the id for us.

Standard identifiers should be used whenever possible. The identifiers can provide some standard graphics or behaviour on some platforms.

ident.h

```
#include <wx/wx.h>
```

```
class Ident : public wxFrame
{
public:
  Ident(const wxString& title);

};
```

ident.cpp

```
#include "ident.h"

Ident::Ident(const wxString& title)
    : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(200,
150))
{

  wxPanel *panel = new wxPanel(this, -1);

  wxGridSizer *grid = new wxGridSizer(2, 3); 改为...(3,2,5,5)

  grid->Add(new wxButton(panel, wxID_CANCEL), 0, wxTOP | wxLEFT, 9);
  grid->Add(new wxButton(panel, wxID_DELETE), 0, wxTOP, 9);
  grid->Add(new wxButton(panel, wxID_SAVE), 0, wxLEFT, 9);
  grid->Add(new wxButton(panel, wxID_EXIT));
  grid->Add(new wxButton(panel, wxID_STOP), 0, wxLEFT, 9);
  grid->Add(new wxButton(panel, wxID_NEW));

  panel->SetSizer(grid);
  Centre();
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
```

```
};
```

main.cpp

```
#include "main.h"
#include "ident.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{
    Ident *ident = new Ident(wxT("Identifiers"));
    ident->Show(true);

    return true;
}
```

In our example we use standard identifiers on buttons. On Linux, the buttons have small icons.

Figure: Identifiers

In this chapter, we talked about events in wxWidgets.

Dialogs

Dialog windows or dialogs are an indispensable part of most modern GUI applications. A dialog is defined as a conversation between two or more persons. In a computer application a dialog is a window which is used to "talk" to the application. A dialog is used to input data, modify data, change the application settings etc. Dialogs are important means of communication between a user and a computer program.

There are essentially two types of dialogs. Predefined dialogs and custom dialogs.

Predefined dialogs

Predefined dialogs are dialogs available in the wxWidgets toolkit. These are dialogs for common programming tasks like showing text, receiving input , loading and saving files etc. They save programmer's time and enhance using some standard behaviour.

Message dialogs

Message dialogs are used to show messages to the user. They are customizable. We can change icons and buttons that will be shown in a dialog.

Messages.h

```
#include <wx/wx.h>

class Messages : public wxFrame
{
public:
    Messages(const wxString& title);

    void ShowMessage1(wxCommandEvent & event);
    void ShowMessage2(wxCommandEvent & event);
    void ShowMessage3(wxCommandEvent & event);
    void ShowMessage4(wxCommandEvent & event);
};
```

```
const int ID_INFO = 1;
const int ID_ERROR = 2;
const int ID_QUESTION = 3;
const int ID_ALERT = 4;
```

Messages.cpp

```
#include "Messages.h"

Messages::Messages(const wxString& title)
       : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(210,
110))
{

  wxPanel *panel = new wxPanel(this, wxID_ANY);

  wxBoxSizer *hbox = new wxBoxSizer(wxHORIZONTAL);
  wxGridSizer *gs = new wxGridSizer(2, 2, 2, 2);

  wxButton *btn1 = new wxButton(panel, ID_INFO, wxT("Info"));
  wxButton *btn2 = new wxButton(panel, ID_ERROR, wxT("Error"));
  wxButton *btn3 = new wxButton(panel, ID_QUESTION, wxT("Question"));
  wxButton *btn4 = new wxButton(panel, ID_ALERT, wxT("Alert"));

  Connect(ID_INFO, wxEVT_COMMAND_BUTTON_CLICKED,
      wxCommandEventHandler(Messages::ShowMessage1));
  Connect(ID_ERROR, wxEVT_COMMAND_BUTTON_CLICKED,
      wxCommandEventHandler(Messages::ShowMessage2));
  Connect(ID_QUESTION, wxEVT_COMMAND_BUTTON_CLICKED,
      wxCommandEventHandler(Messages::ShowMessage3));
  Connect(ID_ALERT, wxEVT_COMMAND_BUTTON_CLICKED,
      wxCommandEventHandler(Messages::ShowMessage4));

  gs->Add(btn1, 1, wxEXPAND);
  gs->Add(btn2, 1);
  gs->Add(btn3, 1);
  gs->Add(btn4, 1);

  hbox->Add(gs, 0, wxALL, 15);
  panel->SetSizer(hbox);
```

```
  Center();
}


void Messages::ShowMessage1(wxCommandEvent& event)
{
  wxMessageDialog *dial = new wxMessageDialog(NULL,
      wxT("Download completed"), wxT("Info"), wxOK);
  dial->ShowModal();
}

void Messages::ShowMessage2(wxCommandEvent& event)
{
   wxMessageDialog *dial = new wxMessageDialog(NULL,
      wxT("Error loading file"), wxT("Error"), wxOK | wxICON_ERROR);
   dial->ShowModal();
}

void Messages::ShowMessage3(wxCommandEvent& event)
{
  wxMessageDialog *dial = new wxMessageDialog(NULL,
      wxT("Are you sure to quit?"), wxT("Question"),
      wxYES_NO | wxNO_DEFAULT | wxICON_QUESTION);
  dial->ShowModal();
}

void Messages::ShowMessage4(wxCommandEvent& event)
{
  wxMessageDialog *dial = new wxMessageDialog(NULL,
      wxT("Unallowed operation"), wxT("Exclamation"),
      wxOK | wxICON_EXCLAMATION);
  dial->ShowModal();
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
```

```
};
```

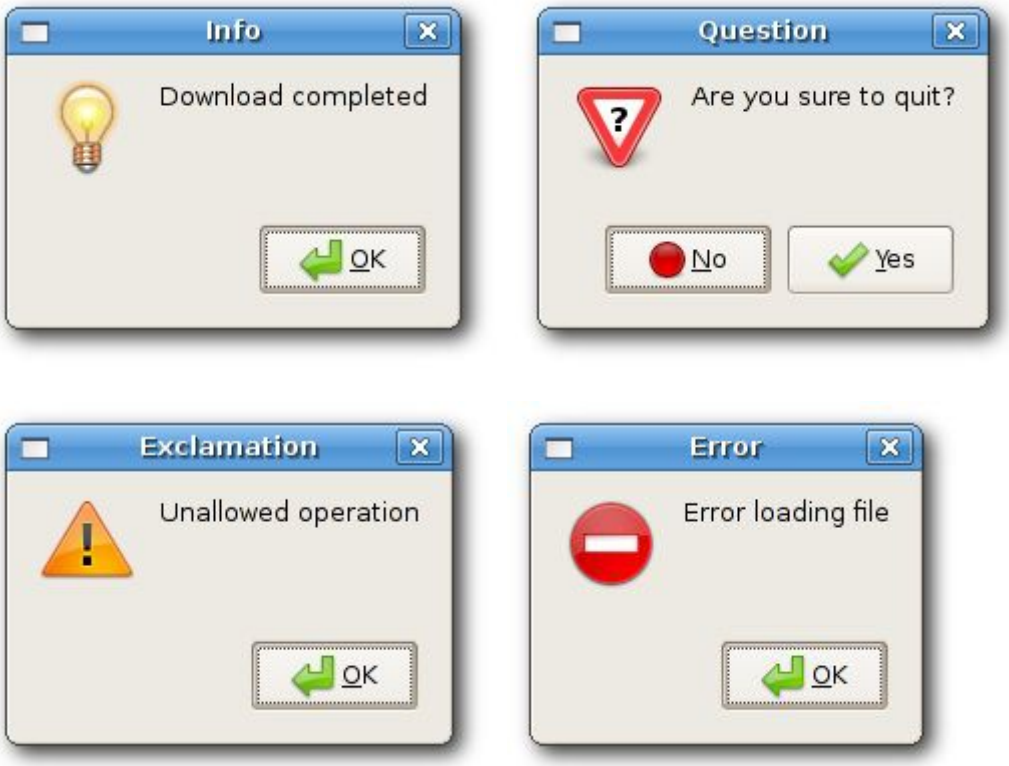
main.cpp

```
#include "main.h"
#include "Messages.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{

  Messages *msgs = new Messages(wxT("Messages"));
  msgs->Show(true);

  return true;
}
```

In our example, we have created four buttons and put them in a grid sizer. These buttons will show four different dialog windows. We create them by specifying different style flags.

```
wxMessageDialog *dial = new wxMessageDialog(NULL,
   wxT("Error loading file"), wxT("Error"), wxOK | wxICON_ERROR);
dial->ShowModal();
```

The creation of the message dialog is simple. We set the dialog to be a toplevel window by providing NULL as a parent. The two strings provide the message text and the dialog title. We show an ok button and an error icon by specifying the wxOK and wxICON_ERROR flags. To show the dialog on screen, we call the *ShowModal()* method.



wxFileDialog

This is a common dialog for opening and saving files.

openfile.h

```
#include <wx/wx.h>

class Openfile : public wxFrame
{
public:
  Openfile(const wxString& title);

  void OnOpen(wxCommandEvent& event);

  wxTextCtrl *tc;
};
```

openfile.cpp

```
#include "openfile.h"

Openfile::Openfile(const wxString & title)
       : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(300,
200))
{

  wxMenuBar *menubar = new wxMenuBar;
  wxMenu *file = new wxMenu;
  file->Append(wxID_OPEN, wxT("&Open"));
  menubar->Append(file, wxT("&File"));
  SetMenuBar(menubar);

  Connect(wxID_OPEN, wxEVT_COMMAND_MENU_SELECTED,
      wxCommandEventHandler(Openfile::OnOpen));


  tc = new wxTextCtrl(this, -1, wxT(""), wxPoint(-1, -1),
      wxSize(-1, -1), wxTE_MULTILINE);

  Center();

}

void Openfile::OnOpen(wxCommandEvent& event)
{

  wxFileDialog * openFileDialog = new wxFileDialog(this);

  if (openFileDialog->ShowModal() == wxID_OK){
      wxString fileName = openFileDialog->GetPath();
      tc->LoadFile(fileName);
  }
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
```

```
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "openfile.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{

    Openfile *open = new Openfile(wxT("Openfile"));
    open->Show(true);

    return true;
}
```

In our example, we display a open file menu item and a simple multiline text control. If we click on the open file menu item a *wxFileDialog* is displayed. We can load some simple text files into the text control.

```
tc = new wxTextCtrl(this, -1, wxT(""), wxPoint(-1, -1),
    wxSize(-1, -1), wxTE_MULTILINE);
```

We load text files into this text control.

```
wxFileDialog * openFileDialog = new wxFileDialog(this);
```

Here we create a wxFileDialog. We use the default parameters. (The open file dialog is the default dialog.)

```
if (openFileDialog->ShowModal() == wxID_OK){
    wxString fileName = openFileDialog->GetPath();
    tc->LoadFile(fileName);
}
```

Here we show the dialog. We get the selected file name and load the file into the text control.

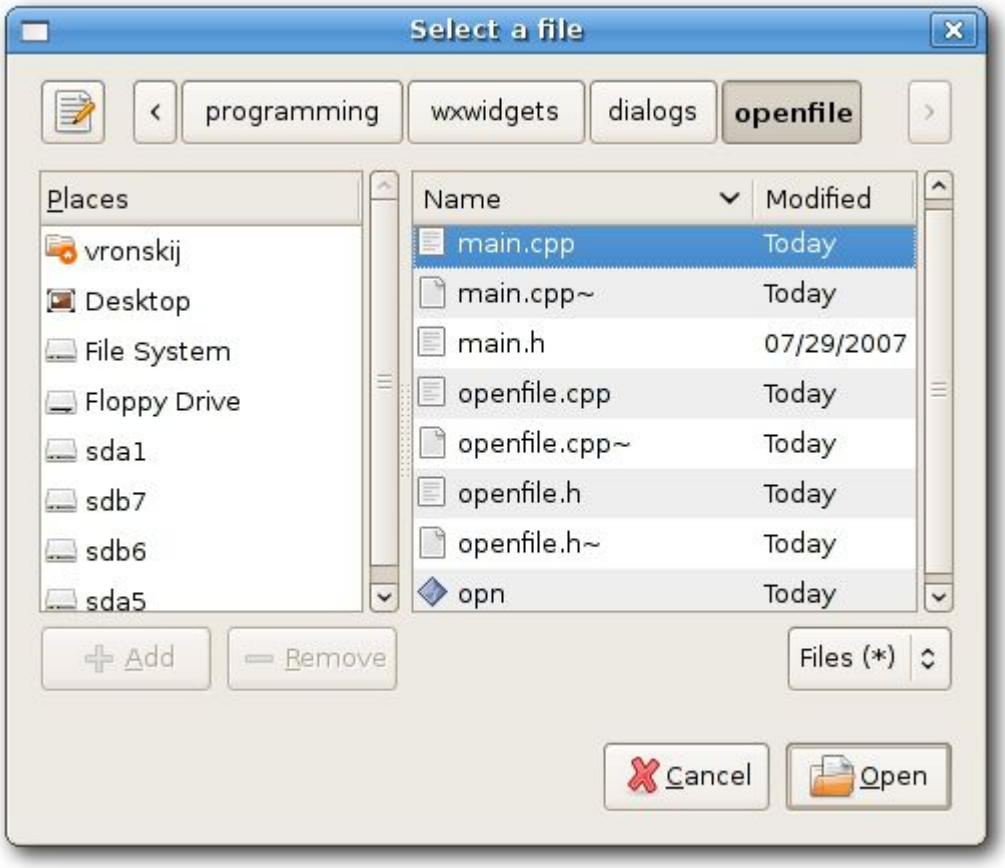


Figure: wxFileDialog on Linux

wxFontDialog

This is a common dialog for choosing a font.

fontdialog.h

```
#include <wx/wx.h>

class ChangeFont : public wxFrame
{
public:
  ChangeFont(const wxString& title);

  void OnOpen(wxCommandEvent& event);
```

```
  wxStaticText *st;

};

const int ID_FONTDIALOG = 1;
```

fontdialog.cpp

```
#include <wx/fontdlg.h>
#include "fontdialog.h"

ChangeFont::ChangeFont(const wxString & title)
       : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(300,
200))
{

  wxPanel *panel = new wxPanel(this, -1);

  wxMenuBar *menubar = new wxMenuBar;
  wxMenu *file = new wxMenu;
  file->Append(ID_FONTDIALOG, wxT("&Change font"));
  menubar->Append(file, wxT("&File"));
  SetMenuBar(menubar);

  Connect(ID_FONTDIALOG, wxEVT_COMMAND_MENU_SELECTED,
      wxCommandEventHandler(ChangeFont::OnOpen));

  st = new wxStaticText(panel, wxID_ANY, wxT("The Agoge"),
      wxPoint(20, 20));

  Center();
}

void ChangeFont::OnOpen(wxCommandEvent& WXUNUSED(event))
{
  wxFontDialog *fontDialog = new wxFontDialog(this);

  if (fontDialog->ShowModal() == wxID_OK) {
    st->SetFont(fontDialog->GetFontData().GetChosenFont());
  }
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "fontdialog.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{

    ChangeFont *change = new ChangeFont(wxT("Change font"));
    change->Show(true);

    return true;
}
```

In this example, we will change the font of a static text example.

```
st = new wxStaticText(panel, wxID_ANY, wxT("The Agoge"),
    wxPoint(20, 20));
```

Here we display a static text on the panel. We will change it's font using the *wxFontDialog*.

```
wxFontDialog *fontDialog = new wxFontDialog(this);

if (fontDialog->ShowModal() == wxID_OK) {
  st->SetFont(fontDialog->GetFontData().GetChosenFont());
}
```

In these code lines, we show the font dialog. Then we get the choosen font. And finally, we change the font of the static text, we created earlier.

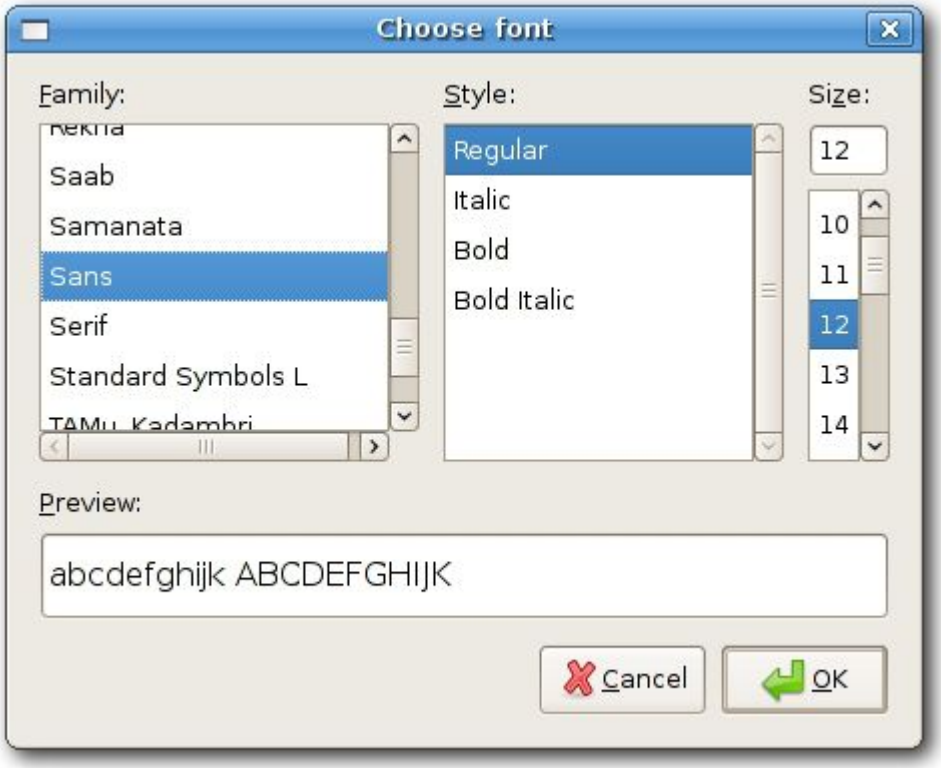


Figure: Font dialog

A custom dialog

In the next example we create a custom dialog. An image editing application can change a color depth of a picture. To provide this funcionality, we could create a suitable custom dialog.

customdialog.h

```
#include <wx/wx.h>

class CustomDialog : public wxDialog
{
public:
  CustomDialog(const wxString& title);

};
```

customdialog.cpp

```
#include "customdialog.h"

CustomDialog::CustomDialog(const wxString & title)
       : wxDialog(NULL, -1, title, wxDefaultPosition, wxSize(250, 230))
{

  wxPanel *panel = new wxPanel(this, -1);

  wxBoxSizer *vbox = new wxBoxSizer(wxVERTICAL);
  wxBoxSizer *hbox = new wxBoxSizer(wxHORIZONTAL);

  wxStaticBox *st = new wxStaticBox(panel, -1, wxT("Colors"),
      wxPoint(5, 5), wxSize(240, 150));
  wxRadioButton *rb = new wxRadioButton(panel, -1,
      wxT("256 Colors"), wxPoint(15, 30), wxDefaultSize, wxRB_GROUP);

  wxRadioButton *rb1 = new wxRadioButton(panel, -1,
      wxT("16 Colors"), wxPoint(15, 55));
  wxRadioButton *rb2 = new wxRadioButton(panel, -1,
      wxT("2 Colors"), wxPoint(15, 80));
  wxRadioButton *rb3 = new wxRadioButton(panel, -1,
      wxT("Custom"), wxPoint(15, 105));
  wxTextCtrl *tc = new wxTextCtrl(panel, -1, wxT(""),
      wxPoint(95, 105));

  wxButton *okButton = new wxButton(this, -1, wxT("Ok"),
      wxDefaultPosition, wxSize(70, 30));
  wxButton *closeButton = new wxButton(this, -1, wxT("Close"),
      wxDefaultPosition, wxSize(70, 30));

  hbox->Add(okButton, 1);
  hbox->Add(closeButton, 1, wxLEFT, 5);

  vbox->Add(panel, 1);
  vbox->Add(hbox, 0, wxALIGN_CENTER | wxTOP | wxBOTTOM, 10);

  SetSizer(vbox);

  Centre();
  ShowModal();

  Destroy();
```

```
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "customdialog.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{

    CustomDialog *custom = new CustomDialog(wxT("CustomDialog"));
    custom->Show(true);

    return true;
}
```

This example is a dialog based application. We illustrate, how to create a custom dialog.

```
class CustomDialog : public wxDialog
```

A custom dialog is based on the *wxDialog* class.

```
wxStaticBox *st = new wxStaticBox(panel, -1, wxT("Colors"),
    wxPoint(5, 5), wxSize(240, 150));
wxRadioButton *rb = new wxRadioButton(panel, -1,
```

```
    wxT("256 Colors"), wxPoint(15, 30), wxDefaultSize, wxRB_GROUP);
```

Note that *wxStaticBox* widget must be created before the widgets that it contains, and that those widgets should be siblings, not children, of the static box.

```
ShowModal();
Destroy();
```

To show the dialog on the screen, we call the *ShowModal()* method. To clear the dialog from the memory, we call the *Destroy()* method.

Figure: Custom dialog

This part of the wxWidgets tutorial was dedicated to dialogs.

Widgets

In this chapter, we will show small examples of several widgets, available in wxWidgets. Widgets are builing blocks of our applications. wxWidgets consists of a large amout of useful widgets. **Widget** is a basic GUI object by definition. A widget gave wxWidgets toolkit a name. This term is used on UNIX systems. On windows, a widget is usually called a control.

wxCheckBox

wxCheckBox is a widget that has two states. On and Off. It is a box with a label. The label can be set to the right or to the left of the box. If the checkbox is checked, it is represented by a tick in a box. A checkbox can be used to show/hide splashscreen at startup, toggle visibility of a toolbar etc.

checkbox.h

```
#include <wx/wx.h>

class CheckBox : public wxFrame
{
public:
    CheckBox(const wxString& title);

    void OnToggle(wxCommandEvent& event);

    wxCheckBox *m_cb;

};

const int ID_CHECKBOX = 100;
```

checkbox.cpp

```
#include "checkbox.h"

CheckBox::CheckBox(const wxString& title)
      : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(270,
150))
```

```
{
  wxPanel *panel = new wxPanel(this, wxID_ANY);

  m_cb = new wxCheckBox(panel, ID_CHECKBOX, wxT("Show title"),
                        wxPoint(20, 20));
  m_cb->SetValue(true);
  Connect(ID_CHECKBOX, wxEVT_COMMAND_CHECKBOX_CLICKED,
          wxCommandEventHandler(CheckBox::OnToggle));
  Centre();
}

void CheckBox::OnToggle(wxCommandEvent& WXUNUSED(event))
{

  if (m_cb->GetValue()) {
      this->SetTitle(wxT("CheckBox"));
  } else {
      this->SetTitle(wxT(" "));
  }
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "checkbox.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{
    CheckBox *cb = new CheckBox(wxT("CheckBox"));
```

```
    cb->Show(true);

    return true;
}
```

In our example, we display one checkbox on the window. We toggle the title of the window by clicking on the checkbox.

```
m_cb = new wxCheckBox(panel, ID_CHECKBOX, wxT("Show title"),
                  wxPoint(20, 20));
m_cb->SetValue(true);
```

We create a checkbox. By default, the title is visible. So we check the checkbox by calling the method *SetValue()*.

```
Connect(ID_CHECKBOX, wxEVT_COMMAND_CHECKBOX_CLICKED,
       wxCommandEventHandler(CheckBox::OnToggle));
```

If we click on the checkbox, a *wxEVT_COMMAND_CHECKBOX_CLICKED* event is generated. We connect this event to the user defined *OnToggle()* method.

```
if (m_cb->GetValue()) {
    this->SetTitle(wxT("CheckBox"));
} else {
    this->SetTitle(wxT(" "));
}
```

Inside the OnToggle() method, we check the state of the checkbox. If it is checked, we display "CheckBox" string in the titlebar, otherwise we clear the title.

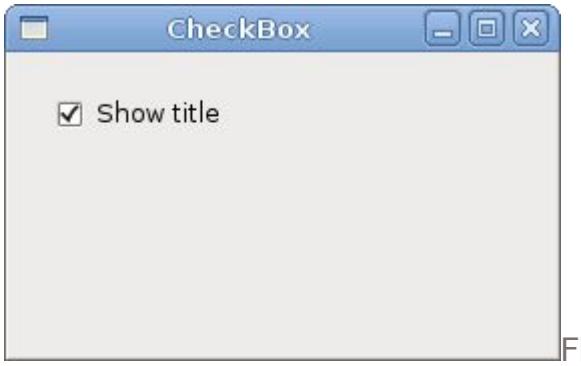


Figure: wxCheckBox

wxBitmapButton

A bitmap button is a button, that displays a bitmap. A bitmap button can have three other states. Selected, focused and displayed. We can set a specific bitmap for those states.

bitmapbutton.h

```
#include <wx/wx.h>
#include <wx/slider.h>

class BitmapButton : public wxFrame
{
public:
  BitmapButton(const wxString& title);

  wxSlider *slider;
  wxBitmapButton *button;
  int pos;

  void OnScroll(wxScrollEvent& event);
};

const int ID_SLIDER = 100;
```

bitmapbutton.cpp

```
#include "bitmapbutton.h"

BitmapButton::BitmapButton(const wxString& title)
      : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(250,
```

```
130))
{
  wxImage::AddHandler( new wxPNGHandler );
  wxPanel *panel = new wxPanel(this);
  slider = new wxSlider(panel, ID_SLIDER, 0, 0, 100,
      wxPoint(10, 30), wxSize(140, -1));

  button = new wxBitmapButton(panel, wxID_ANY,
wxBitmap(wxT("mute.png"),
      wxBITMAP_TYPE_PNG), wxPoint(180, 20));

  Connect(ID_SLIDER, wxEVT_COMMAND_SLIDER_UPDATED,
      wxScrollEventHandler(BitmapButton::OnScroll));
  Center();
}

void BitmapButton::OnScroll(wxScrollEvent& event)
{
  pos = slider->GetValue();

  if (pos == 0) {
      button->SetBitmapLabel(wxBitmap(wxT("mute.png"),
wxBITMAP_TYPE_PNG));
  } else if (pos > 0 && pos <= 30 ) {
      button->SetBitmapLabel(wxBitmap(wxT("min.png"),
wxBITMAP_TYPE_PNG));
  } else if (pos > 30 && pos < 80 ) {
      button->SetBitmapLabel(wxBitmap(wxT("med.png"),
wxBITMAP_TYPE_PNG));
  } else {
      button->SetBitmapLabel(wxBitmap(wxT("max.png"),
wxBITMAP_TYPE_PNG));
  }
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
```

```
};
```

main.cpp

```
#include "main.h"
#include "bitmapbutton.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{

    BitmapButton *bb = new BitmapButton(wxT("BitmapButton"));
    bb->Show(true);

    return true;
}
```

In our example, we have a slider and a bitmap button. We simulate a volume control. By dragging the handle of a slider, we change a bitmap on the button.

```
wxImage::AddHandler( new wxPNGHandler );
```

We will use PNG images, so we must initialize a PNG image handler.

```
button = new wxBitmapButton(panel, wxID_ANY, wxBitmap(wxT("mute.png"),
    wxBITMAP_TYPE_PNG), wxPoint(180, 20));
```

We create a bitmap button. We specify a bitmap type, in our case *wxBITMAP_TYPE_PNG*

```
pos = slider->GetValue();
```

We get the slider value. Depending on this value, we set a bitmap for our button. We have four volume states. Mute, minimum, medium and maximum.

To change a bitmap on the button, we call the *SetBitmapLabel()* method.

Figure: wxBitmapButton

wxToggleButton

wxToggleButton is a button that has two states. Pressed and not pressed. You toggle between these two states by clicking on it. There are situations where this functionality fits well.

togglebutton.h

```
#include <wx/wx.h>
#include <wx/tglbtn.h>

class ToggleButton : public wxFrame
{
public:
  ToggleButton(const wxString& title);

  void OnToggleRed(wxCommandEvent& event);
  void OnToggleGreen(wxCommandEvent& event);
  void OnToggleBlue(wxCommandEvent& event);

protected:
  wxToggleButton *m_tgbutton1;
  wxToggleButton *m_tgbutton2;
  wxToggleButton *m_tgbutton3;

  wxPanel *m_panel;
  wxColour *colour;
};
```

```
const int ID_TGBUTTON1 = 101;
const int ID_TGBUTTON2 = 102;
const int ID_TGBUTTON3 = 103;
```

togglebutton.cpp

```
#include "togglebutton.h"

ToggleButton::ToggleButton(const wxString& title)
      : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(280,
180))
{
  wxPanel *panel = new wxPanel(this, wxID_ANY);

  colour = new wxColour(0, 0, 0);

  m_tgbutton1 = new wxToggleButton(panel, ID_TGBUTTON1,
                                   wxT("Red"), wxPoint(20, 20));
  m_tgbutton2 = new wxToggleButton(panel, ID_TGBUTTON2,
                                   wxT("Green"), wxPoint(20, 70));
  m_tgbutton3 = new wxToggleButton(panel, ID_TGBUTTON3,
                                   wxT("Blue"), wxPoint(20, 120));

  Connect(ID_TGBUTTON1, wxEVT_COMMAND_TOGGLEBUTTON_CLICKED,
      wxCommandEventHandler(ToggleButton::OnToggleRed));
  Connect(ID_TGBUTTON2, wxEVT_COMMAND_TOGGLEBUTTON_CLICKED,
      wxCommandEventHandler(ToggleButton::OnToggleGreen));
  Connect(ID_TGBUTTON3, wxEVT_COMMAND_TOGGLEBUTTON_CLICKED,
      wxCommandEventHandler(ToggleButton::OnToggleBlue));

  m_panel = new wxPanel(panel, wxID_NEW, wxPoint(150, 20),
                        wxSize(110, 110), wxSUNKEN_BORDER);
  m_panel->SetBackgroundColour(colour->GetAsString());
}

void ToggleButton::OnToggleRed(wxCommandEvent& WXUNUSED(event))
{
  unsigned char green = colour->Green();
  unsigned char blue = colour->Blue();

  if ( colour->Red() ) {
      colour->Set(0, green, blue);
```

```
  } else {
      colour->Set(255, green, blue);
  }

  m_panel->SetBackgroundColour(colour->GetAsString());
}

void ToggleButton::OnToggleGreen(wxCommandEvent& WXUNUSED(event))
{
  unsigned char red = colour->Red();
  unsigned char blue = colour->Blue();

  if ( colour->Green() ) {
      colour->Set(red, 0, blue);
  } else {
      colour->Set(red, 255, blue);
  }

  m_panel->SetBackgroundColour(colour->GetAsString());

}

void ToggleButton::OnToggleBlue(wxCommandEvent& WXUNUSED(event))
{
  unsigned char red = colour->Red();
  unsigned char green = colour->Green();

  if ( colour->Blue() ) {
      colour->Set(red, green, 0);

  } else {
      colour->Set(red, green, 255);
  }

  m_panel->SetBackgroundColour(colour->GetAsString());
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
```

```
{
  public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "togglebutton.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{
    ToggleButton *button = new ToggleButton(wxT("ToggleButton"));

    button->Centre();
    button->Show(true);

    return true;
}
```

In our example, we show three toggle buttons and a panel. We set the background color of the panel to black. The togglebuttons will toggle the red, green and blue parts of the color value. The background color will depend on which togglebuttons we have pressed.

```
colour = new wxColour(0, 0, 0);
```

This is the initial color value. No red, green and blue equals to black. Theoretically speaking, black is not a color after all.

```
m_tgbutton1 = new wxToggleButton(panel, ID_TGBUTTON1,
                                 wxT("Red"), wxPoint(20, 20));
```

Here we create a toggle button.

```
Connect(ID_TGBUTTON1, wxEVT_COMMAND_TOGGLEBUTTON_CLICKED,
    wxCommandEventHandler(ToggleButton::OnToggleRed));
```

If we click on the toggle button, a *wxEVT_COMMAND_TOGGLEBUTTON_CLICKED* event is generated. We connect the event handlers for this event. Notice, that we don't connect events to the button methods, but to the *wxFrame*. widget, which is a grand parent of the toggle buttons. It is possible to do this, because command events are propagated to their parents. In our case, button -> panel -> frame. If we wanted to connect the event to the button, we would have to create our derived button classe, which would mean more work.

```
if ( colour->Blue() ) {
    colour->Set(red, green, 0);

} else {
    colour->Set(red, green, 255);
}
```

In the event handlers, we set the respective *wxColour* parameters.

```
m_panel->SetBackgroundColour(colour->GetAsString());
```

We set the background of the panel.

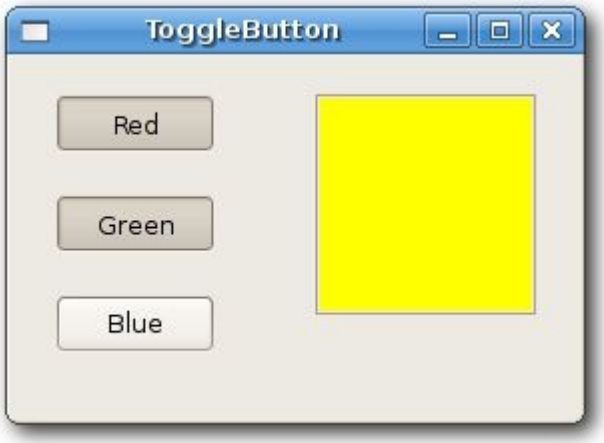


Figure: wxToggleButton

wxStaticLine

This widget displays a simple line on the window. It can be horizontal or vertical.

staticline.h

```
#include <wx/wx.h>

class Staticline : public wxDialog
{
public:
    Staticline(const wxString& title);
};
```

staticline.cpp

```
#include "staticline.h"
#include <wx/stattext.h>
#include <wx/statline.h>

Staticline::Staticline(const wxString& title) : wxDialog(NULL, wxID_ANY,
title,
    wxDefaultPosition, wxSize(360, 350))
{
  wxFont font(10, wxDEFAULT, wxNORMAL, wxBOLD);
  wxStaticText *heading = new wxStaticText(this, wxID_ANY, wxT("The
Central Europe"),
      wxPoint(30, 15));
  heading->SetFont(font);

  wxStaticLine *sl1 = new wxStaticLine(this, wxID_ANY, wxPoint(25, 50),
      wxSize(300,1));

  wxStaticText *st1 = new wxStaticText(this, wxID_ANY, wxT("Slovakia"),
      wxPoint(25, 80));
  wxStaticText *st2 = new wxStaticText(this, wxID_ANY, wxT("Hungary"),
      wxPoint(25, 100));
  wxStaticText *st3 = new wxStaticText(this, wxID_ANY, wxT("Poland"),
      wxPoint(25, 120));
  wxStaticText *st4 = new wxStaticText(this, wxID_ANY, wxT("Czech
```

```
Republic"),
      wxPoint(25, 140));
  wxStaticText *st5 = new wxStaticText(this, wxID_ANY, wxT("Germany"),
      wxPoint(25, 160));
  wxStaticText *st6 = new wxStaticText(this, wxID_ANY, wxT("Slovenia"),
      wxPoint(25, 180));
  wxStaticText *st7 = new wxStaticText(this, wxID_ANY, wxT("Austria"),
      wxPoint(25, 200));
  wxStaticText *st8 = new wxStaticText(this, wxID_ANY,
wxT("Switzerland"),
      wxPoint(25, 220));

  wxStaticText *st9  = new wxStaticText(this, wxID_ANY, wxT("5 379
000"),
      wxPoint(220, 80), wxSize(90, -1), wxALIGN_RIGHT);
  wxStaticText *st10 = new wxStaticText(this, wxID_ANY, wxT("10 084
000"),
      wxPoint(220, 100), wxSize(90, -1), wxALIGN_RIGHT);
  wxStaticText *st11 = new wxStaticText(this, wxID_ANY, wxT("38 635
000"),
      wxPoint(220, 120), wxSize(90, -1), wxALIGN_RIGHT);
  wxStaticText *st12 = new wxStaticText(this, wxID_ANY, wxT("10 240
000"),
      wxPoint(220, 140), wxSize(90, -1), wxALIGN_RIGHT);
  wxStaticText *st13 = new wxStaticText(this, wxID_ANY, wxT("82 443
000"),
      wxPoint(220, 160), wxSize(90, -1), wxALIGN_RIGHT);
  wxStaticText *st14 = new wxStaticText(this, wxID_ANY, wxT("2 001 000"),
      wxPoint(220, 180), wxSize(90, -1), wxALIGN_RIGHT);
  wxStaticText *st15 = new wxStaticText(this, wxID_ANY, wxT("8 032 000"),
      wxPoint(220, 200), wxSize(90, -1), wxALIGN_RIGHT);
  wxStaticText *st16 = new wxStaticText(this, wxID_ANY, wxT("7 288 000"),
      wxPoint(220, 220), wxSize(90, -1), wxALIGN_RIGHT);

  wxStaticLine *sl2 = new wxStaticLine(this, wxID_ANY, wxPoint(25, 260),
      wxSize(300, 1));

  wxStaticText *sum = new wxStaticText(this, wxID_ANY, wxT("164 102
000"),
      wxPoint(220, 280));
  wxFont sum_font = sum->GetFont();
  sum_font.SetWeight(wxBOLD);
  sum->SetFont(sum_font);
```

```
  this->Centre();
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "staticline.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{
    Staticline *sl = new Staticline(wxT("The Central Europe"));
    sl->ShowModal();
    sl->Destroy();

    return true;
}
```

In the previous example, we show Centreal European countries and their populations. The use of*wxStaticLine* makes it more visually attractive.

```
wxStaticLine *sl1 = new wxStaticLine(this, wxID_ANY, wxPoint(25, 50),
   wxSize(300,1));
```

Here we create a horizontal static line. It is 300px wide. The height is 1px.

Figure: wxStaticLine

wxStaticText

A wxStaticText widget displays one or more lines of read-only text.

statictext.h

```
#include <wx/wx.h>

class StaticText : public wxFrame
{
public:
  StaticText(const wxString& title);

};
```

statictext.cpp

```
#include "statictext.h"

StaticText::StaticText(const wxString& title)
      : wxFrame(NULL, wxID_ANY, title)
```

```
{

  wxPanel *panel = new wxPanel(this, wxID_ANY);
  wxString  text = wxT("'Cause sometimes you feel tired,\n\
feel weak, and when you feel weak,\
you feel like you wanna just give up.\n\
But you gotta search within you,\
you gotta find that inner strength\n\
and just pull that shit out of you\
and get that motivation to not give up\n\
and not be a quitter,\
no matter how bad you wanna just fall flat on your face and collapse.");


  wxStaticText *st = new wxStaticText(panel, wxID_ANY, text,
      wxPoint(10, 10), wxDefaultSize, wxALIGN_CENTRE);

  this->SetSize(600, 110);
  this->Centre();
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "statictext.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{
```

```
  StaticText *st = new StaticText(wxT("StaticText"));
  st->Show(true);

  return true;
}
```

In our example, we display a part of the Eminem's Till I Collapse lyrics on the window.

```
wxStaticText *st = new wxStaticText(panel, wxID_ANY, text,
   wxPoint(10, 10), wxDefaultSize, wxALIGN_CENTRE);
```

Here we create the *wxStaticText* widget. The static text is aligned to the cetre.

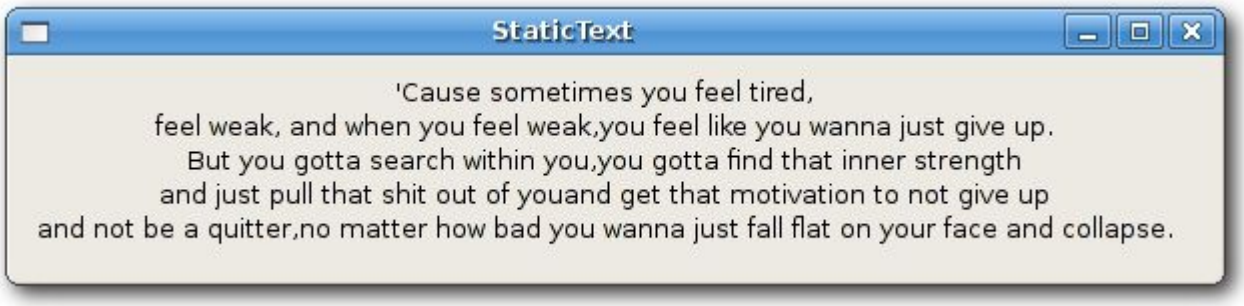


Figure: wxStaticText

wxSlider

wxSlider is a widget that has a simple handle. This handle can be pulled back and forth. This way we are choosing a value for a specific task. Sometimes using a slider is more natural, than simply providing a number or using a spin control.

Slider.h

```
#include <wx/wx.h>
#include <wx/slider.h>

class MyPanel : public wxPanel
{
```

```
public:
    MyPanel(wxFrame *parent);

    void OnPaint(wxPaintEvent& event);
    void OnScroll(wxScrollEvent& event);

    wxSlider *slider;
    int fill;

};

class Slider : public wxFrame
{
public:
    Slider(const wxString& title);

    MyPanel *panel;

};

const int ID_SLIDER = 100;
```

Slider.cpp

```
#include "Slider.h"

Slider::Slider(const wxString& title)
       : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition,
       wxSize(270, 200))
{
  panel = new MyPanel(this);
  Center();
}

MyPanel::MyPanel(wxFrame * parent)
       : wxPanel(parent, wxID_ANY)
{
  fill = 0;
  slider = new wxSlider(this, ID_SLIDER, 0, 0, 140, wxPoint(50, 30),
      wxSize(-1, 140), wxSL_VERTICAL);

  Connect(ID_SLIDER, wxEVT_COMMAND_SLIDER_UPDATED,
```

```
      wxScrollEventHandler(MyPanel::OnScroll));
  Connect(wxEVT_PAINT, wxPaintEventHandler(MyPanel::OnPaint));
}

void MyPanel::OnScroll(wxScrollEvent& event)
{
  fill = slider->GetValue();
  Refresh();
}

void MyPanel::OnPaint(wxPaintEvent& event)
{
  wxPaintDC dc(this);

  wxPen pen(wxColour(212, 212, 212));
  dc.SetPen(pen);

  dc.DrawRectangle(wxRect(140, 30, 80, 140));

  wxBrush brush1(wxColour(197, 108, 0));
  dc.SetBrush(brush1);

  dc.DrawRectangle(wxRect(140, 30, 80, fill));
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "Slider.h"

IMPLEMENT_APP(MyApp)
```

```
bool MyApp::OnInit()
{

    Slider *slider = new Slider(wxT("Slider"));
    slider->Show(true);

    return true;
}
```

In our example, we display a slider widget. By pulling the handle of the slider, we control the background color of the panel. In such an example, using slider is more natural than using e.g. a spin control.

```
slider = new wxSlider(this, ID_SLIDER, 0, 0, 140, wxPoint(50, 30),
   wxSize(-1, 140), wxSL_VERTICAL);
```

We create a vertical slider. The initial value is 0, minimal value is 0 and maximal value is 140. We display no ticks and no labels.

```
Connect(ID_SLIDER, wxEVT_COMMAND_SLIDER_UPDATED,
   wxScrollEventHandler(MyPanel::OnScroll));
```

Here we connect a *wxEVT_COMMAND_SLIDER_UPDATED* event to the *OnScroll()* user defined method.

```
Connect(wxEVT_PAINT, wxPaintEventHandler(MyPanel::OnPaint));
```

We will also do some drawing, so we connect *OnPaint()* method to the *wxEVT_PAINT* event.

```
fill = slider->GetValue();
Refresh();
```

In the *OnScroll()* method, we will get the current slider value. We call the *Refresh()* method, which will generate a *wxEVT_PAINT* event.

```
dc.DrawRectangle(wxRect(140, 30, 80, 140));
...
dc.DrawRectangle(wxRect(140, 30, 80, fill));
```

Inside the *OnPaint()* event handler, we draw two rectangles. The first method is draws a white rectangle with a gray border. The second method draws the a rectangle with some brownish color. The height of the rectangle is controled by the *fill* value, which is set by the slider widget.

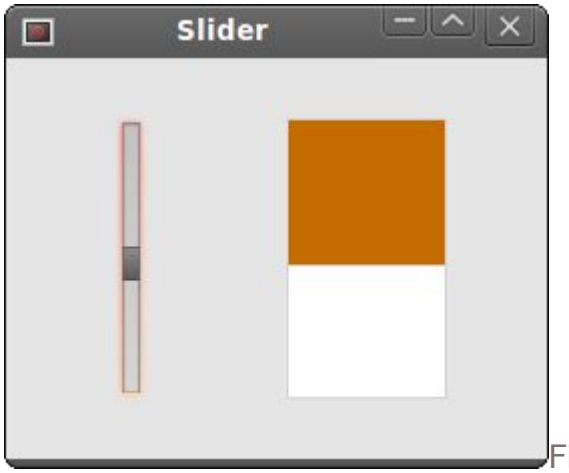


Figure: wxSlider

In this part of the wxWidgets tutorial, we covered various widgets.

Widgets II

In this chapter, we will continue introducing various other widgets. We will mention wxListBox,wxNotebook and wxScrolledWindow.

wxListBox

A wxListBox widget is used for displaying and working with a list of items. As it's name indicates, it is a rectangle that has a list of strings inside. We could use it for displaying a list of mp3 files, book names, module names of a larger project or names of our friends. A wxListBox can be created in two different states. In a single selection state or a multiple selection state. The single selection state is the default state. There are two significant events in wxListBox. The first one is the wxEVT_COMMAND_LISTBOX_SELECTED event. This event is generated when we select a string in a wxListBox. The second one is the wxEVT_COMMAND_LISTBOX_DOUBLE_CLICKED event. It is generated when we double click an item in a wxListBox. The number of elements inside a wxListBox is limited on GTK platform. According to the documentation, it is currently around 2000 elements. The elements are numbered from zero. Scrollbars are displayed automatically if needed.

Listbox.h

```
#include <wx/wx.h>
#include <wx/listbox.h>

class MyPanel : public wxPanel
{
public:
    MyPanel(wxPanel *parent);

    void OnNew(wxCommandEvent& event);
    void OnRename(wxCommandEvent& event);
    void OnClear(wxCommandEvent& event);
    void OnDelete(wxCommandEvent& event);

    wxListBox *m_lb;
```

```
    wxButton *m_newb;
    wxButton *m_renameb;
    wxButton *m_clearb;
    wxButton *m_deleteb;
};

class Listbox : public wxFrame
{
public:
    Listbox(const wxString& title);

    void OnDblClick(wxCommandEvent& event);

    wxListBox *listbox;
    MyPanel *btnPanel;

};

const int ID_RENAME = 1;
const int ID_LISTBOX = 5;
```

Listbox.cpp

```
#include "listbox.h"
#include <wx/textdlg.h>

Listbox::Listbox(const wxString& title)
       : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(270,
200))
{
  wxPanel * panel = new wxPanel(this, -1);

  wxBoxSizer *hbox = new wxBoxSizer(wxHORIZONTAL);

  listbox = new wxListBox(panel, ID_LISTBOX,
      wxPoint(-1, -1), wxSize(-1, -1));

  hbox->Add(listbox, 3, wxEXPAND | wxALL, 20);

  btnPanel = new MyPanel(panel);
  hbox->Add(btnPanel, 2, wxEXPAND | wxRIGHT, 10);
```

```
  Connect(wxEVT_COMMAND_LISTBOX_DOUBLECLICKED,
      wxCommandEventHandler(Listbox::OnDblClick));

  panel->SetSizer(hbox);
  Center();
}

MyPanel::MyPanel(wxPanel * parent)
       : wxPanel(parent, wxID_ANY)
{
  wxBoxSizer *vbox = new wxBoxSizer(wxVERTICAL);

  Listbox *lb = (Listbox *) parent->GetParent();
  m_lb = lb->listbox;

  m_newb = new wxButton(this, wxID_NEW, wxT("New"));
  m_renameb = new wxButton(this, ID_RENAME, wxT("Rename"));
  m_deleteb = new wxButton(this, wxID_DELETE, wxT("Delete"));
  m_clearb = new wxButton(this, wxID_CLEAR, wxT("Clear"));

  Connect(wxID_NEW, wxEVT_COMMAND_BUTTON_CLICKED,
      wxCommandEventHandler(MyPanel::OnNew) );
  Connect(ID_RENAME, wxEVT_COMMAND_BUTTON_CLICKED,
      wxCommandEventHandler(MyPanel::OnRename) );
  Connect(wxID_CLEAR, wxEVT_COMMAND_BUTTON_CLICKED,
      wxCommandEventHandler(MyPanel::OnClear) );
  Connect(wxID_DELETE, wxEVT_COMMAND_BUTTON_CLICKED,
      wxCommandEventHandler(MyPanel::OnDelete) );

  vbox->Add(-1, 20);
  vbox->Add(m_newb);
  vbox->Add(m_renameb, 0, wxTOP, 5);
  vbox->Add(m_deleteb, 0, wxTOP, 5);
  vbox->Add(m_clearb, 0, wxTOP, 5);

  SetSizer(vbox);
}

void MyPanel::OnNew(wxCommandEvent& event)
{
  wxString str = wxGetTextFromUser(wxT("Add new item"));
  if (str.Len() > 0)
      m_lb->Append(str);
}
```

```
void MyPanel::OnClear(wxCommandEvent& event)
{
  m_lb->Clear();
}

void MyPanel::OnRename(wxCommandEvent& event)
{
  wxString text;
  wxString renamed;

  int sel = m_lb->GetSelection();
  if (sel != -1) {
      text = m_lb->GetString(sel);
      renamed = wxGetTextFromUser(wxT("Rename item"),
                 wxT("Rename dialog"), text);
  }

  if (!renamed.IsEmpty()) {
      m_lb->Delete(sel);
      m_lb->Insert(renamed, sel);
  }
}

void MyPanel::OnDelete(wxCommandEvent& event)
{
  int sel = m_lb->GetSelection();
  if (sel != -1) {
      m_lb->Delete(sel);
  }
}

void Listbox::OnDblClick(wxCommandEvent& event)
{
  wxString text;
  wxString renamed;

  int sel = listbox->GetSelection();
  if (sel != -1) {
      text = listbox->GetString(sel);
      renamed = wxGetTextFromUser(wxT("Rename item"),
                 wxT("Rename dialog"), text);
  }
```

```
  if (!renamed.IsEmpty()) {
      listbox->Delete(sel);
      listbox->Insert(renamed, sel);
  }
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "Listbox.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{

    Listbox *listbox = new Listbox(wxT("Listbox"));
    listbox->Show(true);

    return true;
}
```

```
listbox = new wxListBox(panel, ID_LISTBOX,
    wxPoint(-1, -1), wxSize(-1, -1));
```

This is the constructor of the listbox widget.

In our example, we have a list box and four buttons. The buttons are used to add, rename, delete and clear all items in the listbox.

```
wxString str = wxGetTextFromUser(wxT("Add new item"));
if (str.Len() > 0)
    m_lb->Append(str);
```

To add a new string to the listbox, we display a *wxGetTextFromUser* dialog. We call the *Append()*method to append string to the listbox.

```
m_lb->Clear();
```

To clear all items is the easiest action to do. We just call the *Clear()* method.

```
int sel = m_lb->GetSelection();
if (sel != -1) {
    m_lb->Delete(sel);
}
```

To delete an item, we figure out the selected item. Then we call the *Delete()* method.

Renaming an item requires several steps.

```
wxString text;
wxString renamed;
```

We define two local variables.

```
int sel = listbox->GetSelection();
if (sel != -1) {
    text = listbox->GetString(sel);
    renamed = wxGetTextFromUser(wxT("Rename item"),
                wxT("Rename dialog"), text);
}
```

We get the selected string and save it to the renamed variable.

```
if (!renamed.IsEmpty()) {
    m_lb->Delete(sel);
    m_lb->Insert(renamed, sel);
}
```

We check whether the renamed variable is empty. This is to avoid inserting empty strings. Then we delete the old item and insert a new one.

Figure: Listbox

wxNotebook

wxNotebook widget joins multiple windows with corresponding tabs. You can position the Notebook widget using the following style flags:

- wxNB_LEFT
- wxNB_RIGHT
- wxNB_TOP
- wxNB_BOTTOM

The default position is wxNB_TOP.

Notebook.h

```
#include <wx/wx.h>
#include <wx/notebook.h>
```

```
#include <wx/grid.h>

class Notebook : public wxFrame
{
public:
    Notebook(const wxString& title);

    void OnQuit(wxCommandEvent& event);

};

class MyGrid : public wxGrid
{
public:
    MyGrid(wxNotebook *parent);

};
```

Notebook.cpp

```
#include "Notebook.h"

Notebook::Notebook(const wxString& title)
       : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(400,
350))
{
  wxNotebook *nb = new wxNotebook(this, -1, wxPoint(-1, -1),
      wxSize(-1, -1), wxNB_BOTTOM);

  wxMenuBar *menubar = new wxMenuBar;
  wxMenu *file = new wxMenu;
  file->Append(wxID_EXIT, wxT("Quit"), wxT(""));
  menubar->Append(file, wxT("&File"));
  SetMenuBar(menubar);

  Connect(wxEVT_COMMAND_MENU_SELECTED,
      wxCommandEventHandler(Notebook::OnQuit));

  MyGrid *grid1 = new MyGrid(nb);
  MyGrid *grid2 = new MyGrid(nb);
  MyGrid *grid3 = new MyGrid(nb);
```

```
  nb->AddPage(grid1, wxT("Sheet1"));
  nb->AddPage(grid2, wxT("Sheet2"));
  nb->AddPage(grid3, wxT("Sheet3"));

  CreateStatusBar();
  Center();
}

void Notebook::OnQuit(wxCommandEvent& event)
{
  Close(true);
}

MyGrid::MyGrid(wxNotebook * parent)
       : wxGrid(parent, wxID_ANY)
{
  CreateGrid(30, 30);
  SetRowLabelSize(50);
  SetColLabelSize(25);
  SetRowLabelAlignment(wxALIGN_RIGHT, wxALIGN_CENTRE);
  SetLabelFont(wxFont(9, wxFONTFAMILY_DEFAULT,
      wxFONTSTYLE_NORMAL, wxFONTWEIGHT_BOLD));

  for (int i = 0; i < 30 ; i++) {
      this->SetRowSize(i, 25);
  }
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
```

```
#include "Notebook.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{
    Notebook *notebook = new Notebook(wxT("Notebook"));
    notebook->Show(true);

    return true;
}
```

In this example, we have created a notebook widget with three grids. The notebook widget is positioned at the bottom.

```
wxNotebook *nb = new wxNotebook(this, -1, wxPoint(-1, -1),
    wxSize(-1, -1), wxNB_BOTTOM);
```

Here we create the notebook widget.

```
nb->AddPage(grid1, wxT("Sheet1"));
nb->AddPage(grid2, wxT("Sheet2"));
nb->AddPage(grid3, wxT("Sheet3"));
```

We add three grid objects into the notebook widget.

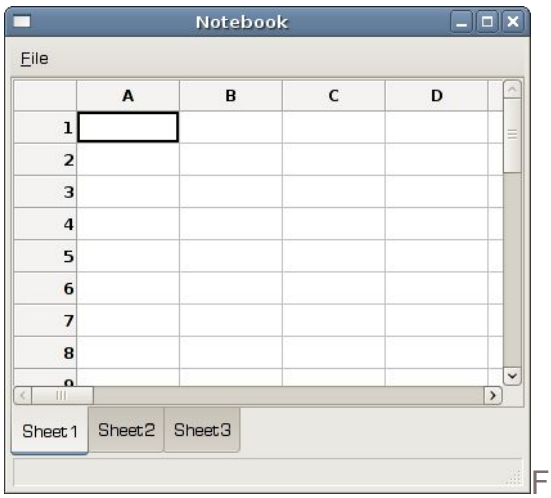


Figure: Notebook widget

wxScrolledWindow

This is one of the container widgets. It can be useful, when we have a larger area than a window can display. In our example, we demonstrate such a case. We place a large image into our window. When the window is smaller than our image, Scrollbars are displayed automatically.

scrolledwindow.h

```
#include <wx/wx.h>

class ScrWindow : public wxFrame
{
public:
  ScrWindow(const wxString& title);

};
```

scrolledwindow.cpp

```
#include "scrolledwindow.h"

ScrWindow::ScrWindow(const wxString& title)
       : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(300,
200))
{
  wxImage::AddHandler(new wxJPEGHandler);
  wxScrolledWindow *sw = new wxScrolledWindow(this);

  wxBitmap bmp(wxT("castle.jpg"), wxBITMAP_TYPE_JPEG);
  wxStaticBitmap *sb = new wxStaticBitmap(sw, -1, bmp);

  int width = bmp.GetWidth();
  int height = bmp.GetHeight();

  sw->SetScrollbars(10, 10, width/10, height/10);
  sw->Scroll(50,10);

  Center();
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "scrolledwindow.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{
    ScrWindow *sw = new ScrWindow(wxT("ScrolledWindow"));
    sw->Show(true);

    return true;
}
```

In our example, we display a picture of a Spis castle.

```
wxImage::AddHandler(new wxJPEGHandler);
```

To handle jpg images, we must initiate the *wxJPEGHandler*.

```
wxScrolledWindow *sw = new wxScrolledWindow(this);

wxBitmap bmp(wxT("castle.jpg"), wxBITMAP_TYPE_JPEG);
wxStaticBitmap *sb = new wxStaticBitmap(sw, -1, bmp);
```

We create a scroll window and put a static bitmap into it.

```
sw->SetScrollbars(10, 10, width/10, height/10);
```

We set the scrollbars.

```
sw->Scroll(50,10);
```

We scroll the window a bit.

In this chapter, we continued covering widgets in wxWidgets library.

Drag and Drop

Wikipedia: In computer graphical user interfaces, drag-and-drop is the action of (or support for the action of) clicking on a virtual object and dragging it to a different location or onto another virtual object. In general, it can be used to invoke many kinds of actions, or create various types of associations between two abstract objects.

Drag and drop functionality is one of the most visible aspects of the graphical user interface. Drag and drop operation enables you to do complex things intuitively.

In drag and drop we basically drag some data from a data source to a data target. So we must have:

- Data object
- Data source
- Data target

For drag & drop of text, wxWidgets has a predefined **wxTextDropTarget** class.

In the following example, we drag and drop file names from the upper list control to the bottom one.

textdrop.h

```
#include <wx/wx.h>
#include <wx/dnd.h>

class TextDrop : public wxFrame
{
public:
  TextDrop(const wxString& title);

  void OnSelect(wxCommandEvent& event);
  void OnDragInit(wxListEvent& event);

  wxGenericDirCtrl *m_gdir;
```

```
  wxListCtrl *m_lc1;
  wxListCtrl *m_lc2;

};

class MyTextDropTarget : public wxTextDropTarget
{
public:
  MyTextDropTarget(wxListCtrl *owner);

  virtual bool OnDropText(wxCoord x, wxCoord y,
      const wxString& data);

  wxListCtrl *m_owner;

};
```

textdrop.cpp

```
#include "textdrop.h"
#include <wx/treectrl.h>
#include <wx/dirctrl.h>
#include <wx/dir.h>
#include <wx/splitter.h>

TextDrop::TextDrop(const wxString& title)
       : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(300,
200))
{

  wxSplitterWindow *spl1 = new wxSplitterWindow(this, -1);
  wxSplitterWindow *spl2 = new wxSplitterWindow(spl1, -1);
  m_gdir = new wxGenericDirCtrl(spl1, -1, wxT("/home/"),
      wxPoint(-1, -1), wxSize(-1, -1), wxDIRCTRL_DIR_ONLY);

  m_lc1 = new wxListCtrl(spl2, -1, wxPoint(-1, -1),
      wxSize(-1, -1), wxLC_LIST);
  m_lc2 = new wxListCtrl(spl2, -1, wxPoint(-1, -1),
      wxSize(-1, -1), wxLC_LIST);

  MyTextDropTarget *mdt = new MyTextDropTarget(m_lc2);
  m_lc2->SetDropTarget(mdt);
```

```
  Connect(m_lc1->GetId(), wxEVT_COMMAND_LIST_BEGIN_DRAG,
      wxListEventHandler(TextDrop::OnDragInit));

  wxTreeCtrl *tree = m_gdir->GetTreeCtrl();

  spl2->SplitHorizontally(m_lc1, m_lc2);
  spl1->SplitVertically(m_gdir, spl2);

  Connect(tree->GetId(), wxEVT_COMMAND_TREE_SEL_CHANGED,
      wxCommandEventHandler(TextDrop::OnSelect));

  Center();
}

MyTextDropTarget::MyTextDropTarget(wxListCtrl *owner)
{
  m_owner = owner;
}

bool MyTextDropTarget::OnDropText(wxCoord x, wxCoord y,
    const wxString& data)
{

  m_owner->InsertItem(0, data);
  return true;

}

void TextDrop::OnSelect(wxCommandEvent& event)
{
  wxString filename;
  wxString path = m_gdir->GetPath();
  wxDir dir(path);

  bool cont = dir.GetFirst(&filename, wxEmptyString, wxDIR_FILES);

  int i = 0;

  m_lc1->ClearAll();
  m_lc2->ClearAll();

  while ( cont )
  {
```

```
      m_lc1->InsertItem(i, filename);
      cont = dir.GetNext(&filename);
      i++;
  }
}

void TextDrop::OnDragInit(wxListEvent& event)
{

 wxString text = m_lc1->GetItemText(event.GetIndex());

 wxTextDataObject tdo(text);
 wxDropSource tds(tdo, m_lc1);
 tds.DoDragDrop(wxDrag_CopyOnly);
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "textdrop.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{

    TextDrop *td = new TextDrop(wxT("TextDrop"));
    td->Show(true);

    return true;
```

```
}
```

In our example, we have a window separated into three parts. This is done by the*wxSplitterWindow* widget. In the left part of the window, we have a generic directory control. We display all directories available under our filesystem. In the right part there are two windows. The first displays all files under a selected directory. The second one is used for dragging the files.

```
MyTextDropTarget *mdt = new MyTextDropTarget(m_lc2);
m_lc2->SetDropTarget(mdt);
```

Here we define a text drop target.

```
wxString text = m_lc1->GetItemText(event.GetIndex());

wxTextDataObject tdo(text);
wxDropSource tds(tdo, m_lc1);
tds.DoDragDrop(wxDrag_CopyOnly);
```

In the *OnDragInit()* method, we define a text data object and a drop source object. We call the*DoDragDrop()* method. The *wxDrag_CopyOnly* constant allows only copying of data.

```
bool MyTextDropTarget::OnDropText(wxCoord x, wxCoord y,
    const wxString& data)
{
  m_owner->InsertItem(0, data);
  return true;
}
```

During the dropping operation, we insert the text data into the list control.

Figure: Drag & Drop

In this chapter, we covered drag and drop operations in wxWidgets.

Device Contexts

The **GDI** (**Graphics Device Interface**) is an interface for working with graphics. It is used to interact with graphic devices such as monitor, printer or a file. The GDI allows programmers to display data on a screen or printer without having to be concerned about the details of a particular device. The GDI insulates the programmer from the hardware.

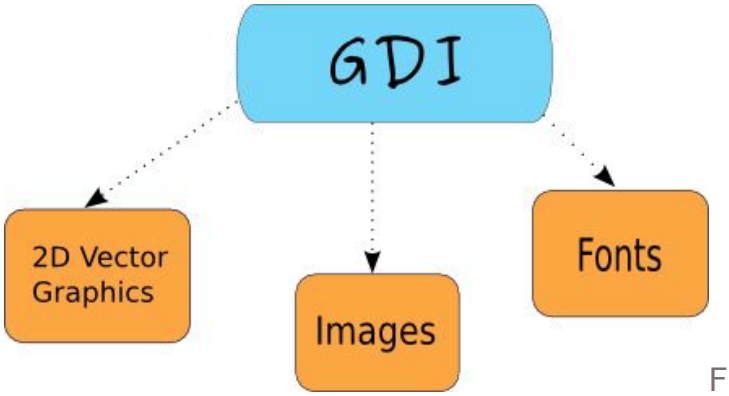


Figure: The GDI structure

From the programmer's point of view, the GDI is a group of classes and methods for working with graphics. The GDI consists of 2D Vector Graphics, Fonts and Images.

To begin drawing graphics, we must create a device context (DC) object. In wxWidgets the device context is called wxDC. The documentation defines wxDC as a device context onto which which graphics and text can be drawn. It represents number of devices in a generic way. Same piece of code can write to different kinds of devices. Be it a screen or a printer. The wxDC is not intended to be used directly. Instead a programmer should choose one of the derived classes. Each derived class is intended to be used under specific conditions.

Derived wxDC classes

- wxBufferedDC
- wxBufferedPaintDC
- wxPostScriptDC
- wxMemoryDC

- wxPrinterDC
- wxScreenDC
- wxClientDC
- wxPaintDC
- wxWindowDC

The wxScreenDC is used to draw anywhere on the screen. The wx.WindowDC is used if we want to paint on the whole window (Windows only). This includes window decorations. The wxClientDC is used to draw on the client area of a window. The client area is the area of a window without it's decorations (title and border). The wxPaintDC is used to draw on the client area as well. But there is one difference between the wxPaintDC and the wxClientDC. The wxPaintDC should be used only from a wxPaintEvent. The wxClientDC shoud not be used from a wxPaintEvent. The wxMemoryDC is used to draw graphics on the bitmap. The wxPostScriptDC is used to write to PostScript files on any platform. The wxPrinterDC is used to access a printer (Windows only).

Simple line

We begin with drawing a line.

line.h

```
#include <wx/wx.h>

class Line : public wxFrame
{
public:
    Line(const wxString& title);

    void OnPaint(wxPaintEvent& event);

};
```

line.cpp

```
#include "line.h"
```

```
Line::Line(const wxString& title)
      : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(280,
180))
{
  this->Connect(wxEVT_PAINT, wxPaintEventHandler(Line::OnPaint));
  this->Centre();
}

void Line::OnPaint(wxPaintEvent& event)
{
  wxPaintDC dc(this);

  wxCoord x1 = 50, y1 = 60;
  wxCoord x2 = 190, y2 = 60;

  dc.DrawLine(x1, y1, x2, y2);
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "line.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{
    Line *line = new Line(wxT("Line"));
    line->Show(true);
```

```
    return true;
}
```

In our example, we draw a simple line onto the client area of the window. If we resize the window, it is redrawn. An *wxPaintEvent* is generated. And the line is drawn again.

```
void OnPaint(wxPaintEvent& event);
```

Here we declare a *OnPaint()* event handler function.

```
this->Connect(wxEVT_PAINT, wxPaintEventHandler(Line::OnPaint));
```

We connect a paint event to the *OnPaint()* method. All the drawing happens inside the *OnPaint()*event handler.

```
wxPaintDC dc(this);
```

We define a *wxPaintDC* device context. It is a device context, that is used to draw on the window inside the *wxPaintEvent*

```
wxCoord x1 = 50, y1 = 60;
wxCoord x2 = 190, y2 = 60;
```

We define four coordinates.

```
  dc.DrawLine(x1, y1, x2, y2);
```

We draw a simple line calling the *DrawLine()* method.

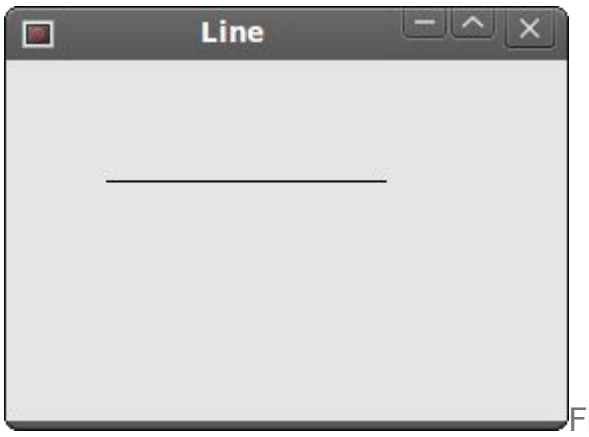


Figure: A simple line

Drawing text

Drawing some text on the window is easy.

text.h

```
#include <wx/wx.h>

class Text : public wxFrame
{
public:
    Text(const wxString & title);

    void OnPaint(wxPaintEvent & event);

};
```

text.cpp

```
#include "text.h"

Text::Text(const wxString& title)
       : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(250,
150))
{
  Connect(wxEVT_PAINT, wxPaintEventHandler(Text::OnPaint));
  Centre();
}
```

```
void Text::OnPaint(wxPaintEvent& event)
{
  wxPaintDC dc(this);

  dc.DrawText(wxT("Лев Николаевич Толстой"), 40, 60);
  dc.DrawText(wxT("Анна Каренина"), 70, 80);
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "text.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{
    Text *text = new Text(wxT("Text"));
    text->Show(true);

    return true;
}
```

In our example, we draw text Lev Nikolayevich Tolstoy, Anna Karenina in Russian azbuka onto the window.

```
  dc.DrawText(wxT("Лев Николаевич Толстой"), 40, 60);
  dc.DrawText(wxT("Анна Каренина"), 70, 80);
```

The *DrawText()* method draws text on the window. It Draws a text string at the specified point, using the current text font, and the current text foreground and background colours. Thanks to the *wxT()* macro, we can use azbuka directly in the code. The wxT() macro is identical to *_T()*macro. It wraps string literals for use with or without Unicode. When Unicode is not enabled, wxT() is an empty macro. When Unicode is enabled, it adds the necessary L for the string literal to become a wide character string constant.

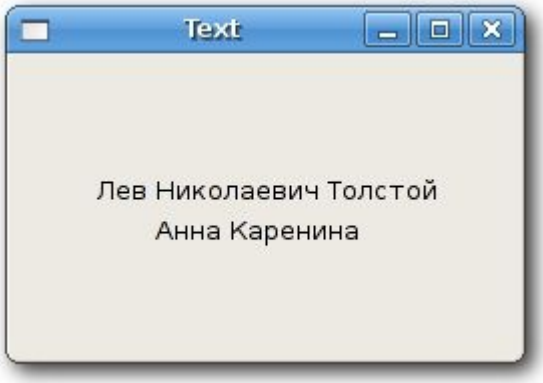


Figure: Drawing text

Point

The simplest geometrical object is a point. It is a plain dot on the window.

```
DrawPoint(int x, int y)
```

This method draws a point at x, y coordinates.

point.h

```
#include <wx/wx.h>

class Points : public wxFrame
{
public:
    Points(const wxString & title);

    void OnPaint(wxPaintEvent & event);
```

```
};
```

points.cpp

```
#include "points.h"
#include <stdlib.h>
#include <time.h>

Points::Points(const wxString& title)
      : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(280,
180))
{

  this->Connect(wxEVT_PAINT, wxPaintEventHandler(Points::OnPaint));
  srand(time(NULL));
  this->Centre();
}

void Points::OnPaint(wxPaintEvent & event)
{
  wxPaintDC dc(this);

  wxCoord x = 0;
  wxCoord y = 0;

  wxSize size = this->GetSize();

  for (int i = 0; i<1000; i++) {
      x = rand() % size.x + 1;
      y = rand() % size.y + 1;
      dc.DrawPoint(x, y);
  }
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
```

```
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "points.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{
    Points *points = new Points(wxT("Points"));
    points->Show(true);

    return true;
}
```

A single point might be difficult to see. So we create 1000 points. Each time the window is resized, we draw the 1000 points over the client area of the window.

```
wxSize size = this->GetSize();
```

Here we get the size of the window.

```
x = rand() % size.x + 1;
```

Here we get a random number in the range of 1 to size.x.

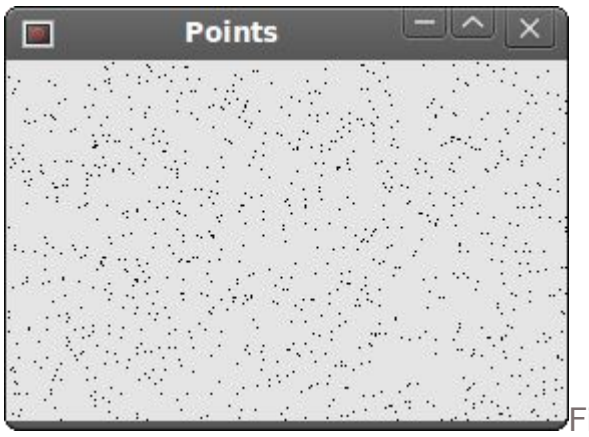


Figure: Points

Pen

Pen is an elementary graphics object. It is used to draw lines, curves and outlines of rectangles, ellipses, polygons or other shapes.

```
wxPen(const wxColour& colour, int width = 1, int style = wxSOLID)
```

The wxPen constructor has three parameters. Colour, width and style. Follows a list of possible pen styles. Pen styles

- wxSOLID
- wxDOT
- wxLONG_DASH
- wxSHORT_DASH
- wxDOT_DASH
- wxTRANSPARENT

pen.h

```
#include <wx/wx.h>

class Pen : public wxFrame
{
public:
    Pen(const wxString& title);

    void OnPaint(wxPaintEvent& event);
```

```
};
```

pen.cpp

```
#include "pen.h"

Pen::Pen(const wxString& title)
       : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(360,
180))
{
  this->Connect(wxEVT_PAINT, wxPaintEventHandler(Pen::OnPaint));
  this->Centre();
}

void Pen::OnPaint(wxPaintEvent& event)
{
  wxPaintDC dc(this);

  wxColour col1, col2;

  col1.Set(wxT("#0c0c0c"));
  col2.Set(wxT("#000000"));

  wxBrush brush(wxColour(255, 255, 255), wxTRANSPARENT);
  dc.SetBrush(brush);

  dc.SetPen(wxPen(col1, 1, wxSOLID));
  dc.DrawRectangle(10, 15, 90, 60);

  dc.SetPen(wxPen(col1, 1, wxDOT));
  dc.DrawRectangle(130, 15, 90, 60);

  dc.SetPen(wxPen(col1, 1, wxLONG_DASH));
  dc.DrawRectangle(250, 15, 90, 60);

  dc.SetPen(wxPen(col1, 1, wxSHORT_DASH));
  dc.DrawRectangle(10, 105, 90, 60);

  dc.SetPen(wxPen(col1, 1, wxDOT_DASH));
  dc.DrawRectangle(130, 105, 90, 60);

  dc.SetPen(wxPen(col1, 1, wxTRANSPARENT));
```

```
  dc.DrawRectangle(250, 105, 90, 60);
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "pen.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{
  Pen *pen = new Pen(wxT("Pen"));
  pen->Show(true);

  return true;
}
```

In our example, we draw 6 rectangles with different pen styles. The last one is transparent, not visible.

```
 dc.SetPen(wxPen(col1, 1, wxSOLID));
 dc.DrawRectangle(10, 15, 90, 60);
```

Here we define a pen for our first rectangle. We set a pen with color col1 (#ocococ), 1 pixel wide, solid. The *DrawRectangle()* method draws the rectangle.

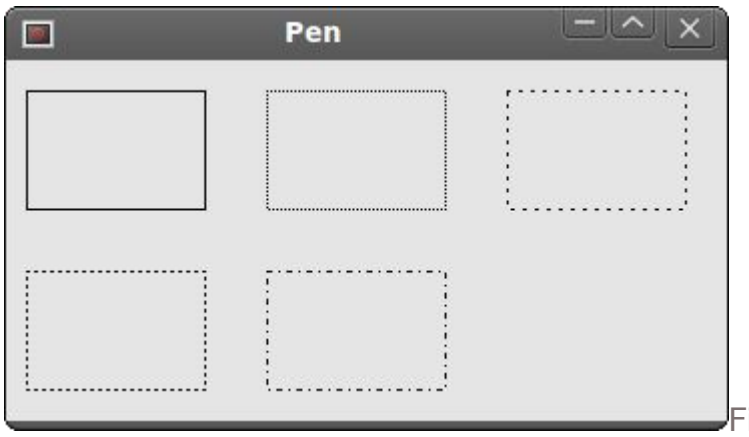


Figure: Pen

Regions

Regions can be combined to create more complex shapes. We can use four set operations. *Union*,*Intersect*, *Substract* and *Xor*. The following example shows all four operations in action.

Regions.h

```
#include <wx/wx.h>

class Regions : public wxFrame
{
public:
  Regions(const wxString & title);

  void OnPaint(wxPaintEvent & event);
};
```

Regions.cpp

```
#include "Regions.h"

Regions::Regions(const wxString& title)
       : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(270,
220))
{
  this->Connect(wxEVT_PAINT, wxPaintEventHandler(Regions::OnPaint));
  this->Centre();
}
```

```
void Regions::OnPaint(wxPaintEvent & event)
{
  wxPaintDC dc(this);
  wxColour gray, white, red, blue;
  wxColour orange, green, brown;

  gray.Set(wxT("#d4d4d4"));
  white.Set(wxT("#ffffff"));
  red.Set(wxT("#ff0000"));
  orange.Set(wxT("#fa8e00"));
  green.Set(wxT("#619e1b"));
  brown.Set(wxT("#715b33"));
  blue.Set(wxT("#0d0060"));

  dc.SetPen(wxPen(gray));

  dc.DrawRectangle(20, 20, 50, 50);
  dc.DrawRectangle(30, 40, 50, 50);

  dc.SetBrush(wxBrush(white));
  dc.DrawRectangle(100, 20, 50, 50);
  dc.DrawRectangle(110, 40, 50, 50);
  wxRegion region1(100, 20, 50, 50);
  wxRegion region2(110, 40, 50, 50);
  region1.Intersect(region2);
  wxRect rect1 = region1.GetBox();
  dc.SetClippingRegion(region1);
  dc.SetBrush(wxBrush(red));
  dc.DrawRectangle(rect1);
  dc.DestroyClippingRegion();

  dc.SetBrush(wxBrush(white));
  dc.DrawRectangle(180, 20, 50, 50);
  dc.DrawRectangle(190, 40, 50, 50);
  wxRegion region3(180, 20, 50, 50);
  wxRegion region4(190, 40, 50, 50);
  region3.Union(region4);
  dc.SetClippingRegion(region3);
  wxRect rect2 = region3.GetBox();
  dc.SetBrush(wxBrush(orange));
  dc.DrawRectangle(rect2);
  dc.DestroyClippingRegion();
```

```
  dc.SetBrush(wxBrush(white));
  dc.DrawRectangle(20, 120, 50, 50);
  dc.DrawRectangle(30, 140, 50, 50);
  wxRegion region5(20, 120, 50, 50);
  wxRegion region6(30, 140, 50, 50);
  region5.Xor(region6);
  wxRect rect3 = region5.GetBox();
  dc.SetClippingRegion(region5);
  dc.SetBrush(wxBrush(green));
  dc.DrawRectangle(rect3);
  dc.DestroyClippingRegion();

  dc.SetBrush(wxBrush(white));
  dc.DrawRectangle(100, 120, 50, 50);
  dc.DrawRectangle(110, 140, 50, 50);
  wxRegion region7(100, 120, 50, 50);
  wxRegion region8(110, 140, 50, 50);
  region7.Subtract(region8);
  wxRect rect4 = region7.GetBox();
  dc.SetClippingRegion(region7);
  dc.SetBrush(wxBrush(brown));
  dc.DrawRectangle(rect4);
  dc.DestroyClippingRegion();

  dc.SetBrush(white);
  dc.DrawRectangle(180, 120, 50, 50);
  dc.DrawRectangle(190, 140, 50, 50);
  wxRegion region9(180, 120, 50, 50);
  wxRegion region10(190, 140, 50, 50);
  region10.Subtract(region9);
  wxRect rect5 = region10.GetBox();
  dc.SetClippingRegion(region10);
  dc.SetBrush(wxBrush(blue));
  dc.DrawRectangle(rect5);
  dc.DestroyClippingRegion();
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
```

```
{
  public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "Regions.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{
  Regions *regions = new Regions(wxT("Regions"));
  regions->Show(true);

  return true;
}
```

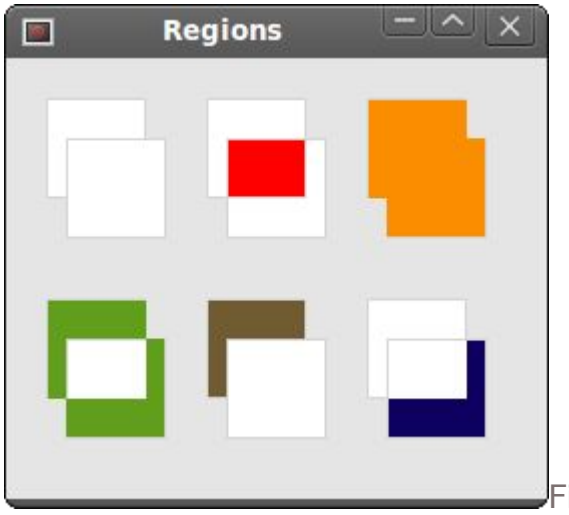


Figure: Regions

Gradient

In computer graphics, gradient is a smooth blending of shades from light to dark or from one color to another. In 2D drawing programs and paint programs, gradients are used to create colorful backgrounds and special effects as well as to simulate lights and shadows. (answers.com)

```
  void GradientFillLinear(const wxRect& rect, const wxColour&
initialColour,
      const wxColour& destColour, wxDirection nDirection = wxEAST)
```

This method fills the area specified by a rect with a linear gradient, starting from initialColour and eventually fading to destColour. The nDirection parameter specifies the direction of the colour change, the default value is wxEAST.

gradient.h

```
#include <wx/wx.h>

class Gradient : public wxFrame
{
public:
  Gradient(const wxString& title);

  void OnPaint(wxPaintEvent& event);

};
```

gradient.cpp

```
#include "gradient.h"

Gradient::Gradient(const wxString& title)
      : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(220,
260))
{
  this->Connect(wxEVT_PAINT, wxPaintEventHandler(Gradient::OnPaint));
  this->Centre();
}

void Gradient::OnPaint(wxPaintEvent& event)
{
  wxPaintDC dc(this);


  wxColour col1, col2;
```

```
  col1.Set(wxT("#e12223"));
  col2.Set(wxT("#000000"));

  dc.GradientFillLinear(wxRect(20, 20, 180, 40), col1, col2, wxNORTH);
  dc.GradientFillLinear(wxRect(20, 80, 180, 40), col1, col2, wxSOUTH);
  dc.GradientFillLinear(wxRect(20, 140, 180, 40), col1, col2, wxEAST);
  dc.GradientFillLinear(wxRect(20, 200, 180, 40), col1, col2, wxWEST);
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "gradient.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{
  Gradient *grad = new Gradient(wxT("Gradient"));
  grad->Show(true);

  return true;
}
```

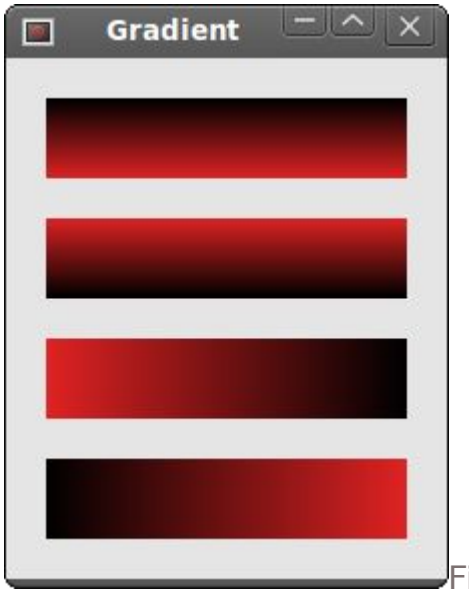


Figure: Gradient

Shapes

Shapes are more sophisticated geometrical objects. We will draw various geometrical shapes in the following example.

shapes.h

```
#include <wx/wx.h>

class Shapes : public wxFrame
{
public:
    Shapes(const wxString & title);

    void OnPaint(wxPaintEvent & event);

};
```

shapes.cpp

```
#include "shapes.h"

Shapes::Shapes(const wxString& title)
       : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(350,
300))
```

```
{
  this->Connect(wxEVT_PAINT, wxPaintEventHandler(Shapes::OnPaint));
  this->Centre();
}

void Shapes::OnPaint(wxPaintEvent& event)
{
  wxPaintDC dc(this);

  wxPoint lines[] = { wxPoint(20, 260), wxPoint(100, 260),
         wxPoint(20, 210), wxPoint(100, 210) };
  wxPoint polygon[] = { wxPoint(130, 140), wxPoint(180, 170),
         wxPoint(180, 140), wxPoint(220, 110), wxPoint(140, 100) };
  wxPoint splines[] = { wxPoint(240, 170), wxPoint(280, 170),
         wxPoint(285, 110), wxPoint(325, 110) };

  dc.DrawEllipse(20, 20, 90, 60);
  dc.DrawRoundedRectangle(130, 20, 90, 60, 10);
  dc.DrawArc(240, 40, 340, 40, 290, 20);

  dc.DrawPolygon(4, polygon);
  dc.DrawRectangle(20, 120, 80, 50);
  dc.DrawSpline(4, splines);

  dc.DrawLines(4, lines);
  dc.DrawCircle(170, 230, 35);
  dc.DrawRectangle(250, 200, 60, 60);
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "shapes.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{

    Shapes *shapes = new Shapes(wxT("Shapes"));
    shapes->Show(true);

    return true;
}
```

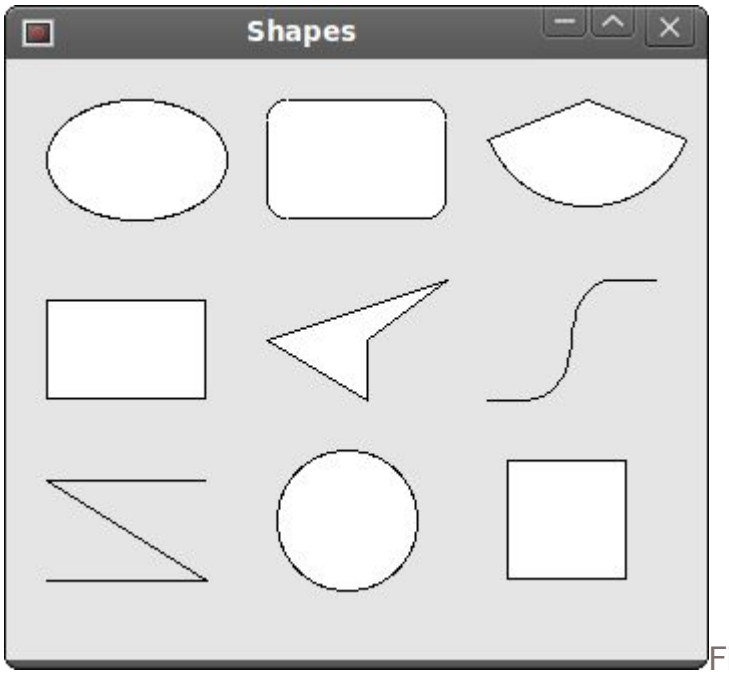


Figure: Shapes

In this chapter, we covered GDI in wxWidgets.

Custom widgets

Have you ever looked at an application and wondered, how a particular GUI item was created? Probably every wannabe programmer has. Then you were looking at a list of widgets provided by your favourite gui library. But you couldn't find it. Toolkits usually provide only the most common widgets like buttons, text widgets, sliders etc. No toolkit can provide all possible widgets.

There are actually two kinds of toolkits. Spartan toolkits and heavy weight toolkits. The FLTK toolkit is a kind of a spartan toolkit. It provides only the very basic widgets and assumes, that the programemer will create the more complicated ones himself. wxWidgets is a heavy weight one. It has lots of widgets. Yet it does not provide the more specialized widgets. For example a speed meter widget, a widget that measures the capacity of a CD to be burned (found e.g. in nero). Toolkits also don't have usually charts.

Programmers must create such widgets by themselves. They do it by using the drawing tools provided by the toolkit. There are two possibilities. A programmer can modify or enhance an existing widget. Or he can create a custom widget from scratch.

Here I assume, you have read the chapter on the Device Contexts.

The Burning Widget

This is an example of a widget, that we create from scratch. This widget can be found in various media burning applications, like Nero Burning ROM.

widget.h

```
#ifndef WIDGET_H
#define WIDGET_H

#include <wx/wx.h>

class Widget : public wxPanel
{
public:
```

```
  Widget(wxPanel *parent, int id );

  wxPanel *m_parent;


  void OnSize(wxSizeEvent& event);
  void OnPaint(wxPaintEvent& event);

};
#endif
```

widget.cpp

```
#include <wx/wx.h>
#include "widget.h"
#include "burning.h"

int num[] = { 75, 150, 225, 300, 375, 450, 525, 600, 675 };
int asize = sizeof(num)/sizeof(num[1]);

Widget::Widget(wxPanel *parent, int id)
      : wxPanel(parent, id, wxDefaultPosition, wxSize(-1, 30),
wxSUNKEN_BORDER)
{

  m_parent = parent;

  Connect(wxEVT_PAINT, wxPaintEventHandler(Widget::OnPaint));
  Connect(wxEVT_SIZE, wxSizeEventHandler(Widget::OnSize));

}

void Widget::OnPaint(wxPaintEvent& event)
{
  wxFont font(9, wxFONTFAMILY_DEFAULT, wxFONTSTYLE_NORMAL,
            wxFONTWEIGHT_NORMAL, false, wxT("Courier 10 Pitch"));

  wxPaintDC dc(this);
  dc.SetFont(font);
  wxSize size = GetSize();
  int width = size.GetWidth();
```

```
  Burning *burn = (Burning *) m_parent->GetParent();

  int cur_width = burn->GetCurWidth();

  int step = (int) round(width / 10.0);

  int till = (int) ((width / 750.0) * cur_width);
  int full = (int) ((width / 750.0) * 700);

  if (cur_width >= 700) {

      dc.SetPen(wxPen(wxColour(255, 255, 184)));
      dc.SetBrush(wxBrush(wxColour(255, 255, 184)));
      dc.DrawRectangle(0, 0, full, 30);
      dc.SetPen(wxPen(wxColour(255, 175, 175)));
      dc.SetBrush(wxBrush(wxColour(255, 175, 175)));
      dc.DrawRectangle(full, 0, till-full, 30);

  } else {

      dc.SetPen(wxPen(wxColour(255, 255, 184)));
      dc.SetBrush(wxBrush(wxColour(255, 255, 184)));
      dc.DrawRectangle(0, 0, till, 30);

  }

  dc.SetPen(wxPen(wxColour(90, 80, 60)));
  for ( int i=1; i <= asize; i++ ) {

  dc.DrawLine(i*step, 0, i*step, 6);
  wxSize size = dc.GetTextExtent(wxString::Format(wxT("%d"),
num[i-1]));
  dc.DrawText(wxString::Format(wxT("%d"), num[i-1]),
      i*step-size.GetWidth()/2, 8);
   }
}

void Widget::OnSize(wxSizeEvent& event)
{
  Refresh();
}
```

burning.h

```
#ifndef BURNING_H
#define BURNING_H

#include <wx/wx.h>
#include "widget.h"

class Burning : public wxFrame
{
public:
  Burning(const wxString& title);

  void OnScroll(wxScrollEvent& event);
  int GetCurWidth();


  wxSlider *m_slider;
  Widget *m_wid;

  int cur_width;

};
#endif
```

burning.cpp

```
#include "burning.h"
#include "widget.h"

int ID_SLIDER = 1;

Burning::Burning(const wxString& title)
       : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(350,
200))
{

  cur_width = 75;

  wxPanel *panel = new wxPanel(this, wxID_ANY);
  wxPanel *centerPanel = new wxPanel(panel, wxID_ANY);
```

```
  m_slider = new wxSlider(centerPanel, ID_SLIDER, 75, 0, 750, wxPoint(-1,
-1),
      wxSize(150, -1), wxSL_LABELS);

  wxBoxSizer *vbox = new wxBoxSizer(wxVERTICAL);
  wxBoxSizer *hbox = new wxBoxSizer(wxHORIZONTAL);
  wxBoxSizer *hbox2 = new wxBoxSizer(wxHORIZONTAL);
  wxBoxSizer *hbox3 = new wxBoxSizer(wxHORIZONTAL);

  m_wid = new Widget(panel, wxID_ANY);
  hbox->Add(m_wid, 1, wxEXPAND);

  hbox2->Add(centerPanel, 1, wxEXPAND);
  hbox3->Add(m_slider, 0, wxTOP | wxLEFT, 35);

  centerPanel->SetSizer(hbox3);

  vbox->Add(hbox2, 1, wxEXPAND);
  vbox->Add(hbox, 0, wxEXPAND);

  panel->SetSizer(vbox);
  m_slider->SetFocus();

  Connect(ID_SLIDER, wxEVT_COMMAND_SLIDER_UPDATED,
      wxScrollEventHandler(Burning::OnScroll));

  Centre();
}

void Burning::OnScroll(wxScrollEvent& WXUNUSED(event))
{
 cur_width = m_slider->GetValue();
 m_wid->Refresh();
}

int Burning::GetCurWidth()
{
 return cur_width;
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
    virtual bool OnInit();
};
```

main.cpp

```
#include "main.h"
#include "burning.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{

    Burning *burning = new Burning(wxT("The Burning Widget"));
    burning->Show(true);

    return true;
}
```

We put a wxPanel on the bottom of the window and draw the entire widget manually. All the important code resides in the OnPaint() method of the Widget class. This widget shows graphically the total capacity of a medium and the free space available to us. The widget is controlled by a slider widget. The minimum value of our custom widget is 0, the maximum is 750. If we reach value 700, we began drawing in red colour. This normally indicates overburning.

```
wxSize size = GetSize();
int width = size.GetWidth();
...
int till = (int) ((width / 750.0) * cur_width);
int full = (int) ((width / 750.0) * 700);
```

We draw the widget dynamically. The greater the window, the greater the

burning widget. And vice versa. That is why we must calculate the size of the wxPanel onto which we draw the custom widget. The till parameter determines the total size to be drawn. This value comes from the slider widget. It is a proportion of the whole area. The full parameter determines the point, where we begin to draw in red color. Notice the use of floating point arithmetics. This is to achieve greater precision.

The actual drawing consists of three steps. We draw the yellow or red and yellow rectangle. Then we draw the vertical lines, which divide the widget into several parts. Finally, we draw the numbers, which indicate the capacity of the medium.

```
void Widget::OnSize(wxSizeEvent& event)
{
  Refresh();
}
```

Every time the window is resized, we refresh the widget. This causes the widget to repaint itself.

```
void Burning::OnScroll(wxScrollEvent& WXUNUSED(event))
{
 cur_width = m_slider->GetValue();
 m_wid->Refresh();
}
```

If we scroll the thumb of the slider, we get the actual value and save it into the *cur_width*variable. This value is used, when the burning widget is drawn. Then we cause the widget to be redrawn.

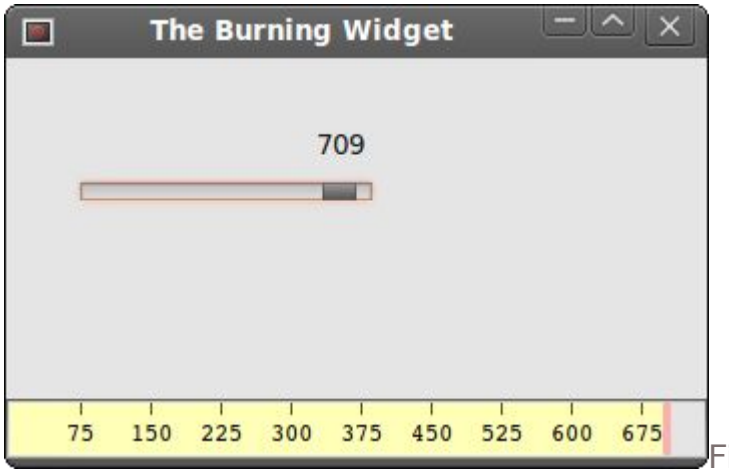


Figure: Burning Widget

In this part of the wxWidgets tutorial, we have created a custom widget.

The Tetris game in wxWidgets

The Tetris game is one of the most popular computer games ever created. The original game was designed and programmed by a Russian programmer **Alexey Pajitnov** in 1985. Since then, tetris is available on almost every computer platform in lots of variations. Even my mobile phone has a modified version of the Tetris game.

Tetris is called a falling block puzzle game. In this game, we have seven different shapes called**tetrominoes**. S-shape, Z-shape, T-shape, L-shape, Line-shape, MirroredL-shape and a Square-shape. Each of these shapes is formed with four squares. The shapes are falling down the board. The object of the tetris game is to move and rotate the shapes, so that they fit as much as possible. If we manage to form a row, the row is destroyed and we score. We play the tetris game until we top out.

Figure: Tetrominoes

wxWidgets is a toolkit designed to create applications. There are other libraries which are targeted at creating computer games. Nevertheless, wxWidgets and other application toolkits can be used to create games.

The development

We do not have images for our tetris game, we draw the tetrominoes using the drawing API available in the wxWidgets programming toolkit. Behind every computer game, there is a mathematical model. So it is in tetris.

Some ideas behind the game.

- We use **wxTimer** to create a game cycle

- The tetrominoes are drawn
- The shapes move on a square by square basis (not pixel by pixel)
- Mathematically a board is a simple list of numbers

Shape.h

```
#ifndef SHAPE_H
#define SHAPE_H

enum Tetrominoes { NoShape, ZShape, SShape, LineShape,
                   TShape, SquareShape, LShape, MirroredLShape };

class Shape
{
public:
    Shape() { SetShape(NoShape); }
    void SetShape(Tetrominoes shape);
    void SetRandomShape();

    Tetrominoes GetShape() const { return pieceShape; }
    int x(int index) const { return coords[index][0]; }
    int y(int index) const { return coords[index][1]; }

    int MinX() const;
    int MaxX() const;
    int MinY() const;
    int MaxY() const;

    Shape RotateLeft() const;
    Shape RotateRight() const;

private:
    void SetX(int index, int x) { coords[index][0] = x; }
    void SetY(int index, int y) { coords[index][1] = y; }
    Tetrominoes pieceShape;
    int coords[4][2];
};

#endif
```

Shape.cpp

```
#include <stdlib.h>
#include <algorithm>
#include "Shape.h"

using namespace std;

void Shape::SetShape(Tetrominoes shape)
{
    static const int coordsTable[8][4][2] = {
        { { 0, 0 },   { 0, 0 },   { 0, 0 },   { 0, 0 } },
        { { 0, -1 },  { 0, 0 },   { -1, 0 },  { -1, 1 } },
        { { 0, -1 },  { 0, 0 },   { 1, 0 },   { 1, 1 } },
        { { 0, -1 },  { 0, 0 },   { 0, 1 },   { 0, 2 } },
        { { -1, 0 },  { 0, 0 },   { 1, 0 },   { 0, 1 } },
        { { 0, 0 },   { 1, 0 },   { 0, 1 },   { 1, 1 } },
        { { -1, -1 }, { 0, -1 },  { 0, 0 },   { 0, 1 } },
        { { 1, -1 },  { 0, -1 },  { 0, 0 },   { 0, 1 } }
    };

    for (int i = 0; i < 4 ; i++) {
        for (int j = 0; j < 2; ++j)
            coords[i][j] = coordsTable[shape][i][j];
    }
    pieceShape = shape;
}

void Shape::SetRandomShape()
{
  int x = rand() % 7 + 1;
  SetShape(Tetrominoes(x));
}

int Shape::MinX() const
{
  int m = coords[0][0];
  for (int i=0; i<4; i++) {
      m = min(m, coords[i][0]);
  }
  return m;
}

int Shape::MaxX() const
```

```
{
  int m = coords[0][0];
  for (int i=0; i<4; i++) {
      m = max(m, coords[i][0]);
  }
  return m;
}

int Shape::MinY() const
{
  int m = coords[0][1];
  for (int i=0; i<4; i++) {
      m = min(m, coords[i][1]);
  }
  return m;
}

int Shape::MaxY() const
{
  int m = coords[0][1];
  for (int i=0; i<4; i++) {
      m = max(m, coords[i][1]);
  }
  return m;
}

Shape Shape::RotateLeft() const
{
    if (pieceShape == SquareShape)
        return *this;

    Shape result;
    result.pieceShape = pieceShape;
    for (int i = 0; i < 4; ++i) {
        result.SetX(i, y(i));
        result.SetY(i, -x(i));
    }
    return result;
}

Shape Shape::RotateRight() const
{
    if (pieceShape == SquareShape)
        return *this;
```

```
    Shape result;
    result.pieceShape = pieceShape;
    for (int i = 0; i < 4; ++i) {
        result.SetX(i, -y(i));
        result.SetY(i, x(i));
    }
    return result;
}
```

Board.h

```
#ifndef BOARD_H
#define BOARD_H

#include "Shape.h"
#include <wx/wx.h>

class Board : public wxPanel
{
public:
    Board(wxFrame *parent);
    void Start();
    void Pause();
    void linesRemovedChanged(int numLines);

protected:
    void OnPaint(wxPaintEvent& event);
    void OnKeyDown(wxKeyEvent& event);
    void OnTimer(wxCommandEvent& event);

private:
    enum { BoardWidth = 10, BoardHeight = 22 };

    Tetrominoes & ShapeAt(int x, int y) { return board[(y * BoardWidth)
+ x]; }

    int SquareWidth() { return GetClientSize().GetWidth() /
BoardWidth; }
    int SquareHeight() { return GetClientSize().GetHeight() /
BoardHeight; }
    void ClearBoard();
```

```
    void DropDown();
    void OneLineDown();
    void PieceDropped();
    void RemoveFullLines();
    void NewPiece();
    bool TryMove(const Shape& newPiece, int newX, int newY);
    void DrawSquare(wxPaintDC &dc, int x, int y, Tetrominoes shape);

    wxTimer *timer;
    bool isStarted;
    bool isPaused;
    bool isFallingFinished;
    Shape curPiece;
    int curX;
    int curY;
    int numLinesRemoved;
    Tetrominoes board[BoardWidth * BoardHeight];
    wxStatusBar *m_stsbar;
};

#endif
```

Board.cpp

```
#include "Board.h"

Board::Board(wxFrame *parent)
      : wxPanel(parent, wxID_ANY, wxDefaultPosition,
            wxDefaultSize, wxBORDER_NONE)
{
    timer = new wxTimer(this, 1);

    m_stsbar = parent->GetStatusBar();
    isFallingFinished = false;
    isStarted = false;
    isPaused = false;
    numLinesRemoved = 0;
    curX = 0;
    curY = 0;

    ClearBoard();
```

```
    Connect(wxEVT_PAINT, wxPaintEventHandler(Board::OnPaint));
    Connect(wxEVT_KEY_DOWN, wxKeyEventHandler(Board::OnKeyDown));
    Connect(wxEVT_TIMER, wxCommandEventHandler(Board::OnTimer));
}

void Board::Start()
{
    if (isPaused)
        return;

    isStarted = true;
    isFallingFinished = false;
    numLinesRemoved = 0;
    ClearBoard();

    NewPiece();
    timer->Start(300);
}

void Board::Pause()
{
    if (!isStarted)
        return;

    isPaused = !isPaused;
    if (isPaused) {
        timer->Stop();
        m_stsbar->SetStatusText(wxT("paused"));
    } else {
        timer->Start(300);
        wxString str;
        str.Printf(wxT("%d"), numLinesRemoved);
        m_stsbar->SetStatusText(str);
    }
    Refresh();
}

void Board::OnPaint(wxPaintEvent& event)
{
    wxPaintDC dc(this);

    wxSize size = GetClientSize();
    int boardTop = size.GetHeight() - BoardHeight * SquareHeight();
```

```
    for (int i = 0; i < BoardHeight; ++i) {
        for (int j = 0; j < BoardWidth; ++j) {
            Tetrominoes shape = ShapeAt(j, BoardHeight - i - 1);
            if (shape != NoShape)
                DrawSquare(dc, 0 + j * SquareWidth(),
                           boardTop + i * SquareHeight(), shape);
        }
    }

    if (curPiece.GetShape() != NoShape) {
        for (int i = 0; i < 4; ++i) {
            int x = curX + curPiece.x(i);
            int y = curY - curPiece.y(i);
            DrawSquare(dc, 0 + x * SquareWidth(),
                       boardTop + (BoardHeight - y - 1) * SquareHeight(),
                       curPiece.GetShape());
        }
    }
}

void Board::OnKeyDown(wxKeyEvent& event)
{
    if (!isStarted || curPiece.GetShape() == NoShape) {
        event.Skip();
        return;
    }

    int keycode = event.GetKeyCode();

    if (keycode == 'p' || keycode == 'P') {
    Pause();
        return;
    }

    if (isPaused)
        return;

    switch (keycode) {
    case WXK_LEFT:
        TryMove(curPiece, curX - 1, curY);
        break;
    case WXK_RIGHT:
        TryMove(curPiece, curX + 1, curY);
```

```
        break;
    case WXK_DOWN:
        TryMove(curPiece.RotateRight(), curX, curY);
        break;
    case WXK_UP:
        TryMove(curPiece.RotateLeft(), curX, curY);
        break;
    case WXK_SPACE:
        DropDown();
        break;
    case 'd':
        OneLineDown();
        break;
    case 'D':
        OneLineDown();
        break;
    default:
        event.Skip();
    }

}

void Board::OnTimer(wxCommandEvent& event)
{
    if (isFallingFinished) {
        isFallingFinished = false;
        NewPiece();
    } else {
        OneLineDown();
    }
}

void Board::ClearBoard()
{
    for (int i = 0; i < BoardHeight * BoardWidth; ++i)
        board[i] = NoShape;
}

void Board::DropDown()
{
    int newY = curY;
    while (newY > 0) {
        if (!TryMove(curPiece, curX, newY - 1))
            break;
```

```
        --newY;
    }
    PieceDropped();
}

void Board::OneLineDown()
{
    if (!TryMove(curPiece, curX, curY - 1))
        PieceDropped();
}

void Board::PieceDropped()
{
    for (int i = 0; i < 4; ++i) {
        int x = curX + curPiece.x(i);
        int y = curY - curPiece.y(i);
        ShapeAt(x, y) = curPiece.GetShape();
    }

    RemoveFullLines();

    if (!isFallingFinished)
        NewPiece();
}

void Board::RemoveFullLines()
{
     int numFullLines = 0;

     for (int i = BoardHeight - 1; i >= 0; --i) {
         bool lineIsFull = true;

         for (int j = 0; j < BoardWidth; ++j) {
             if (ShapeAt(j, i) == NoShape) {
                 lineIsFull = false;
                 break;
             }
         }

         if (lineIsFull) {
             ++numFullLines;
             for (int k = i; k < BoardHeight - 1; ++k) {
                 for (int j = 0; j < BoardWidth; ++j)
                     ShapeAt(j, k) = ShapeAt(j, k + 1);
```

```
            }
        }
    }

    if (numFullLines > 0) {
        numLinesRemoved += numFullLines;
        wxString str;
    str.Printf(wxT("%d"), numLinesRemoved);
        m_stsbar->SetStatusText(str);

        isFallingFinished = true;
        curPiece.SetShape(NoShape);
        Refresh();
    }
 }

void Board::NewPiece()
{
    curPiece.SetRandomShape();
    curX = BoardWidth / 2 + 1;
    curY = BoardHeight - 1 + curPiece.MinY();

    if (!TryMove(curPiece, curX, curY)) {
        curPiece.SetShape(NoShape);
        timer->Stop();
        isStarted = false;
        m_stsbar->SetStatusText(wxT("game over"));
    }
}

bool Board::TryMove(const Shape& newPiece, int newX, int newY)
{
    for (int i = 0; i < 4; ++i) {
        int x = newX + newPiece.x(i);
        int y = newY - newPiece.y(i);
        if (x < 0 || x >= BoardWidth || y < 0 || y >= BoardHeight)
            return false;
        if (ShapeAt(x, y) != NoShape)
            return false;
    }

    curPiece = newPiece;
    curX = newX;
    curY = newY;
```

```
    Refresh();
    return true;
}

void Board::DrawSquare(wxPaintDC& dc, int x, int y, Tetrominoes shape)
{
    static wxColour colors[] = { wxColour(0, 0, 0), wxColour(204, 102,
102),
            wxColour(102, 204, 102), wxColour(102, 102, 204),
            wxColour(204, 204, 102), wxColour(204, 102, 204),
            wxColour(102, 204, 204), wxColour(218, 170, 0) };

    static wxColour light[] = { wxColour(0, 0, 0), wxColour(248, 159,
171),
            wxColour(121, 252, 121), wxColour(121, 121, 252),
            wxColour(252, 252, 121), wxColour(252, 121, 252),
            wxColour(121, 252, 252), wxColour(252, 198, 0) };

    static wxColour dark[] = { wxColour(0, 0, 0), wxColour(128, 59, 59),
            wxColour(59, 128, 59), wxColour(59, 59, 128),
            wxColour(128, 128, 59), wxColour(128, 59, 128),
            wxColour(59, 128, 128), wxColour(128, 98, 0) };

    wxPen pen(light[int(shape)]);
    pen.SetCap(wxCAP_PROJECTING);
    dc.SetPen(pen);

    dc.DrawLine(x, y + SquareHeight() - 1, x, y);
    dc.DrawLine(x, y, x + SquareWidth() - 1, y);

    wxPen darkpen(dark[int(shape)]);
    darkpen.SetCap(wxCAP_PROJECTING);
    dc.SetPen(darkpen);

    dc.DrawLine(x + 1, y + SquareHeight() - 1,
        x + SquareWidth() - 1, y + SquareHeight() - 1);
    dc.DrawLine(x + SquareWidth() - 1,
        y + SquareHeight() - 1, x + SquareWidth() - 1, y + 1);

    dc.SetPen(*wxTRANSPARENT_PEN);
    dc.SetBrush(wxBrush(colors[int(shape)]));
    dc.DrawRectangle(x + 1, y + 1, SquareWidth() - 2,
        SquareHeight() - 2);
```

```
}
```

Tetris.h

```
#include <wx/wx.h>

class Tetris : public wxFrame
{
public:
    Tetris(const wxString& title);

};
```

Tetris.cpp

```
#include "Tetris.h"
#include "Board.h"

Tetris::Tetris(const wxString& title)
       : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(180,
380))
{
  wxStatusBar *sb = CreateStatusBar();
  sb->SetStatusText(wxT("0"));
  Board *board = new Board(this);
  board->SetFocus();
  board->Start();
}
```

main.h

```
#include <wx/wx.h>

class MyApp : public wxApp
{
  public:
    virtual bool OnInit();

};
```

main.cpp

```
#include "main.h"
#include "Tetris.h"

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{
    srand(time(NULL));
    Tetris *tetris = new Tetris(wxT("Tetris"));
    tetris->Centre();
    tetris->Show(true);

    return true;
}
```

I have simplified the game a bit, so that it is easier to understand. The game starts immediately, after it is launched. We can pause the game by pressing the p key. The space key will drop the tetris piece immediately to the bottom. The d key will drop the piece one line down. (It can be used to speed up the falling a bit.) The game goes at constant speed, no acceleration is implemented. The score is the number of lines, that we have removed.

```
...
isFallingFinished = false;
isStarted = false;
isPaused = false;
numLinesRemoved = 0;
curX = 0;
curY = 0;
...
```

Before we start the game, we initialize some important variables. The **isFallingFinished**variable determines, it the tetris shape has finished falling and we then need to create a new shape. The **numLinesRemoved** counts the number of lines, we have removed so far. The **curX**and **curY** variables determine the actual position of the falling tetris shape.

```
for (int i = 0; i < BoardHeight; ++i) {
    for (int j = 0; j < BoardWidth; ++j) {
        Tetrominoes shape = ShapeAt(j, BoardHeight - i - 1);
        if (shape != NoShape)
            DrawSquare(dc, 0 + j * SquareWidth(),
                   boardTop + i * SquareHeight(), shape);
    }
}
```

The painting of the game is divided into two steps. In the first step, we draw all the shapes, or remains of the shapes, that have been dropped to the bottom of the board. All the squares are rememberd in the **board** array. We access it using the **ShapeAt()** method.

```
if (curPiece.GetShape() != NoShape) {
    for (int i = 0; i < 4; ++i) {
        int x = curX + curPiece.x(i);
        int y = curY - curPiece.y(i);
        DrawSquare(dc, 0 + x * SquareWidth(),
                   boardTop + (BoardHeight - y - 1) * SquareHeight(),
                   curPiece.GetShape());
    }
}
```

The next step is drawing of the actual piece, that is falling down.

```
...
switch (keycode) {
case WXK_LEFT:
    TryMove(curPiece, curX - 1, curY);
    break;
...
```

In the **Board::OnKeyDown()** method we check for pressed keys. If we press the left arrow key, we try to move the piece to the left. We say try, because the piece might not be able to move.

```
void Board::OnTimer(wxCommandEvent& event)
```

```
{
    if (isFallingFinished) {
        isFallingFinished = false;
        NewPiece();
    } else {
        OneLineDown();
    }
}
```

In the **Board::OnTimer()** method we either create a new piece, after the previous one was dropped to the bottom, or we move a falling piece one line down.

```
void Board::DropDown()
{
    int newY = curY;
    while (newY > 0) {
        if (!TryMove(curPiece, curX, newY - 1))
            break;
        --newY;
    }
    PieceDropped();
}
```

The **Board::DropDown()** method drops the falling shape immediately to the bottom of the board. It happens, when we press the space key.

```
void Board::PieceDropped()
{
    for (int i = 0; i < 4; ++i) {
        int x = curX + curPiece.x(i);
        int y = curY - curPiece.y(i);
        ShapeAt(x, y) = curPiece.GetShape();
    }

    RemoveFullLines();

    if (!isFallingFinished)
        NewPiece();
```

```
}
```

In the **Board::PieceDropped()** method we set the current shape at it's final position. We call the **RemoveFullLines()** method to check, if we have at least one full line. And we create a new tetris shape, if it wasn't already created in the **Board::PieceDropped()** method in the meantime.

```
 if (lineIsFull) {
     ++numFullLines;
     for (int k = i; k < BoardHeight - 1; ++k) {
         for (int j = 0; j < BoardWidth; ++j)
             ShapeAt(j, k) = ShapeAt(j, k + 1);
     }
 }
```

This code removes the full lines. After finding a full line we increase the counter. We move all the lines above the full row one line down. This way we destroy the full line. Notice, that in our tetris game, we use so called **naive gravity**. This means, that the squares may be left floating above empty gaps.

```
void Board::NewPiece()
{
    curPiece.SetRandomShape();
    curX = BoardWidth / 2 + 1;
    curY = BoardHeight - 1 + curPiece.MinY();

    if (!TryMove(curPiece, curX, curY)) {
        curPiece.SetShape(NoShape);
        timer->Stop();
        isStarted = false;
        m_stsbar->SetStatusText(wxT("game over"));
    }
}
```

The **Board::NewPiece()** method creates randomly a new tetris piece. If the piece cannot go into it's initial position, the game is over.

```
bool Board::TryMove(const Shape& newPiece, int newX, int newY)
```

```
{
    for (int i = 0; i < 4; ++i) {
        int x = newX + newPiece.x(i);
        int y = newY - newPiece.y(i);
        if (x < 0 || x >= BoardWidth || y < 0 || y >= BoardHeight)
            return false;
        if (ShapeAt(x, y) != NoShape)
            return false;
    }

    curPiece = newPiece;
    curX = newX;
    curY = newY;
    Refresh();
    return true;
}
```

In the **Board::TryMove()** method we try to move our shapes. If the shape is at the edge of the board or is adjacent to some other shape, we return false. Otherwise we place the current falling shape to a new position and return true.

The **Shape** class saves information about the tetris piece.

```
for (int i = 0; i < 4 ; i++) {
    for (int j = 0; j < 2; ++j)
        coords[i][j] = coordsTable[shape][i][j];
}
```

The coords array saves the coordinates of the tetris piece. For example, numbers { 0, -1 }, { 0, 0 }, { 1, 0 }, { 1, 1 } , represent a rotated S-shape. The following diagram illustrates the shape.

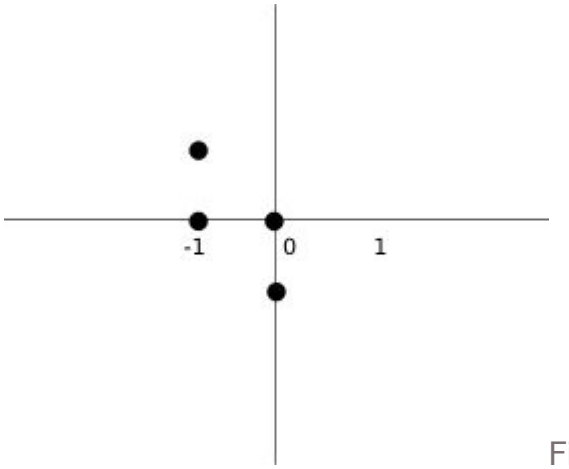


Figure: Coordinates

When we draw the current falling piece, we draw it at **curX**, **curY** position. Then we look at the coordinates table and draw all the four squares.

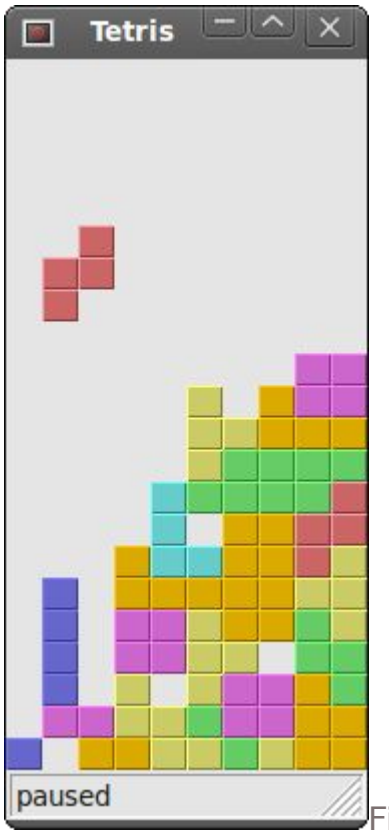


Figure: Tetris

This was a Tetris game in wxWidgets.